VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# INFERENCE PROPOJENÍ KOMPONENT

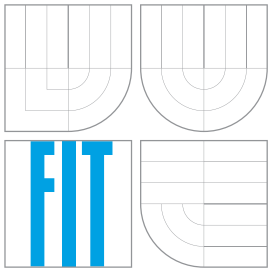
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. NELA OLŠAROVÁ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# INFERENCE PROPOJENÍ KOMPONENT

COMPONENT INTERCONNECTION INFERENCE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. NELA OLŠAROVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZBYNĚK KŘIVKA, Ph.D.

BRNO 2012

# Abstrakt

Tento dokument popisuje řešení diplomové práce zabývající se návrhem algoritmu pro inferenci propojení hardwarových komponent. Algoritmus je určen pro použití v editoru návrhu schémat pro FPGA čipy, který je součástí školního vývojového prostředí VLAM IDE. Algoritmus má uživateli pomoci s nalezením optimálního propojení dvou vybraných komponent. Vývojové prostředí s editorem návrhu je implementováno jako zásuvný modul do prostředí Eclipse, kdy je využit grafický modelovací rámec GMF. Po úvodu do těchto technologií a metod návrhu vestavěných systémů následuje návrh inferenčního algoritmu. Tento problém spadá pod problémy kombinatorické optimalizace, konkrétně je příbuzný s přiřazovacím problémem a bipartitním párováním. Poté je popsána implementace algoritmu a grafického uživatelského rozhraní pro jeho použití, následuje jeho otestování a shrnutí dosažených výsledků.



# Abstract

This document describes works on the master thesis. The thesis deals with the design of hardware component interconnection inference algorithm that is supposed to be used in the FPGA schema editor that was integrated into educational integrated development environment VLAM IDE. The aim of the algorithm is to support user by finding an optimal interconnection of two given components. The editor and the development environment are implemented as an Eclipse plugin using GMF framework. A brief description of this technologies and the embedded systems design are followed by the design of the inference algorithm. This problem is a topic of combinatorial optimization, related to the bipartite matching and assignment problem. After this, the implementation of the algorithm is described, followed by tests and a summary of achieved results.



# Klíčová slova

inference, algoritmus, vestavěný systém, návrh vestavěných systémů, FPGA, jazyk Wright, komponentní modelování, komponentní systém, komponenta, port, rozhraní, kombinatorická optimalizace, přiřazovací problém, bipartitní párování, strom, Eclipse, EMF, GEF, GMF

# Keywords

inference, algorithm, embedded system, embedded system design, FPGA, Wright language, component modelling, component system, component, port, interface, combinatorial optimization, assignment problem, bipartite matching, tree, Eclipse, EMF, GEF, GMF

# Citace

Nela Olšarová: Inference propojení komponent, diplomová práce, Brno, FIT VUT v Brně, 2012

# Inference propojení komponent

## Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracovala samostatně pod vedením Ing. Zbyňka Křivky, PhD. a v seznamu použité literatury jsem uvedla všechny prameny, ze kterých jsem čerpala.

.....
Nela Olšarová
31. července 2012

## Poděkování

Děkuji vedoucímu své práce Ing. Zbyňku Křivkovi, PhD. za odborné vedení, rady, inspiraci a čas strávený na konzultacích. Poděkování patří také Ing. Otovi Jirákovi a Ing. Zdeňku Vašíčkovi, PhD. za jejich připomínky a technické podklady, kterými přispěli k vývoji.

# Obsah

# Úvod

Tato technická zpráva popisuje teoretická východiska, návrh, implementaci a testování výsledků diplomové práce „Inference propojení komponent“ řešené v ak. roce 2011/2012. Obsah zprávy je rozšířením dokumentu, který vznikl jako semestrální projekt v zimním semestru tohoto akademického roku [22]. Výstupem semestrálního projektu byl zejména průzkum teoretických poznatků a z nich vycházející návrh inferenčního algoritmu. Diplomová práce na semestrální projekt navazuje implementací navrženého algoritmu a jeho testováním.

Inferenční algoritmus je určen pro použití v komponentním editoru návrhu pro FPGA, který je součástí integrovaného vývojového prostředí VLAM IDE. Toto vývojové prostředí vzniklo na Fakultě informačních technologií VUT v Brně v rámci řešení projektu „Virtuální laboratoř aplikace mikroprocesorové techniky“ (VLAM) probíhajícího v letech 2006-2011 pod číslem MŠMT 2C06008. V tomto projektu byla část zde popisované diplomové práce využita. Úkolem implementovaného algoritmu je pomoci uživateli při vytváření schémat zapojení, kdy mu po výběru dvou komponent napoví jejich nejlepší propojení.

Prostředí VLAM IDE s editorem návrhů pro FPGA bylo vytvořeno jako zásuvné moduly do rámce Eclipse. Kapitola 1 se proto bude věnovat stručnému popisu tohoto rámce a dalších v projektu využívaných rámců pro modelování a vytváření grafických editorů pod Eclipse. Následovat bude seznámení s vestavěnými systémy včetně metodologie jejich návrhu a komponentního modelování v kapitole 2. Kapitola 3 bude věnována projektu VLAM, a zejména pak popisu integrovaného vývojového prostředí VLAM IDE a editoru návrhu, kde bude inferenční algoritmus ve výsledku nasazen.

Návrh inferenčního algoritmu je založen na poznatcích získaných studiem tzv. přiřazovacích problémů a bipartitního párování, jim tedy bude věnována samostatná kapitola 4. Samotný návrh algoritmu je potom rozebrán v následující kapitole 5, včetně diskuze jeho očekávané úspěšnosti. V závěru této kapitoly bude navrženo uživatelské rozhraní pro aplikaci a úpravu výstupů algoritmu, které bude součástí stávajícího uživatelského rozhraní prostředí VLAM IDE.

Implementace navrženého algoritmu a uživatelského rozhraní pro tento algoritmus bude popsána v kapitole 6. Algoritmus bude dále otestován na reálných aplikacích z výuky a výsledky tohoto testování budou shrnuty v kapitole 7. V závěrečné kapitole budou zrekapitulovány a zhodnoceny výsledky dosažené v této diplomové práci i vzhledem k jejich možnému budoucímu rozšíření a vylepšení.

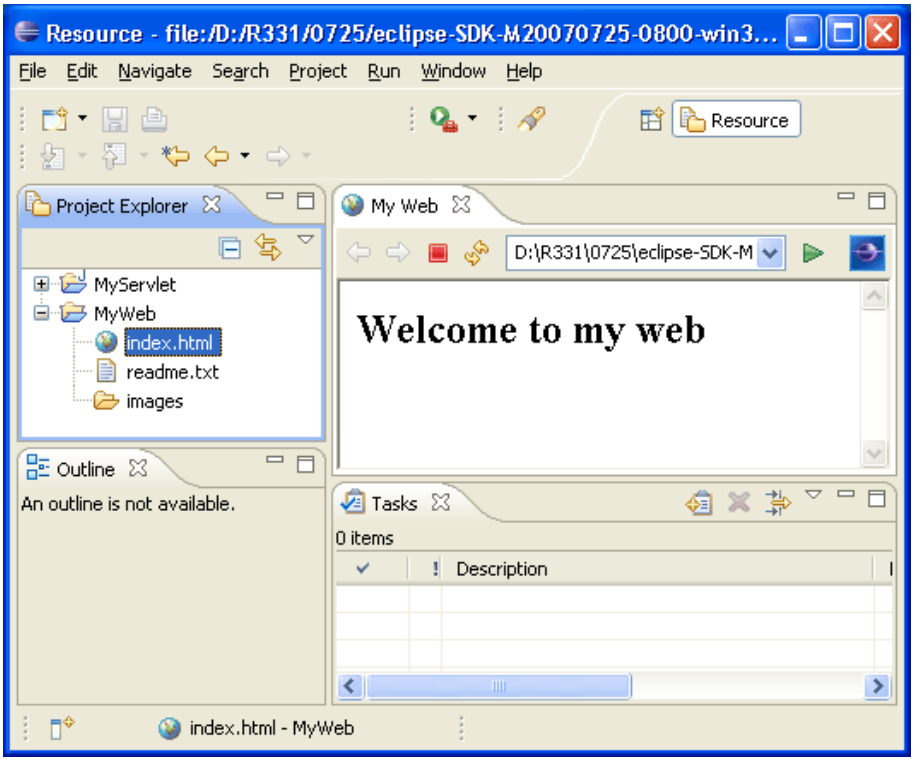
# Kapitola 1

# Platforma Eclipse

Jedná se o vývojovou platformu s otevřeným zdrojovým kódem (*open source*) distribuovanou pod licencí EPL (Eclipse Public License). Platforma Eclipse poskytuje obecné uživatelské rozhraní (UI, *user interface*) a aplikační rozhraní (API) pro běh zásuvných modulů (*plug-in*), které zajišťují koncovému uživateli již specifickou funkcionalitu. Lze ji tedy použít jako univerzální základ pro vývoj vlastních vývojových prostředí (IDE, *Integrated Development Environment*).

Eclipse je multiplatformní, umožňuje tedy běh na více typech operačních systémů (OS), přičemž samotný vývoj zásuvných modulů pro Eclipse je již od OS víceméně odstíněn. Jazykem implementace platformy Eclipse a zároveň všech zásuvných modulů je jazyk Java [30].

Na obr. 1.1 vidíme základní podobu Eclipse bez přidaných zásuvných modulů.




Obrázek 1.1: Základní podoba Eclipse [30]

V dalších částech kapitoly zmíníme modulární koncept platformy a popíšeme její strukturu a součásti. Dále se budeme věnovat rámcům pro vytváření grafických editorů, modelování a vytváření grafických editorů modelů založených na platformě Eclipse.

## 1.1 Koncept zásuvných modulů

Zásuvné moduly rozšiřují funkčnost platformy Eclipse, mohou to být knihovny zdrojových kódů, rozšíření (*extension*) platformy, případně dokumentace. Mohou také samy definovat body rozšíření (*extension point*), tj. místa pro připojení dalších zásuvných modulů poskytujících rozšíření původních modulů.

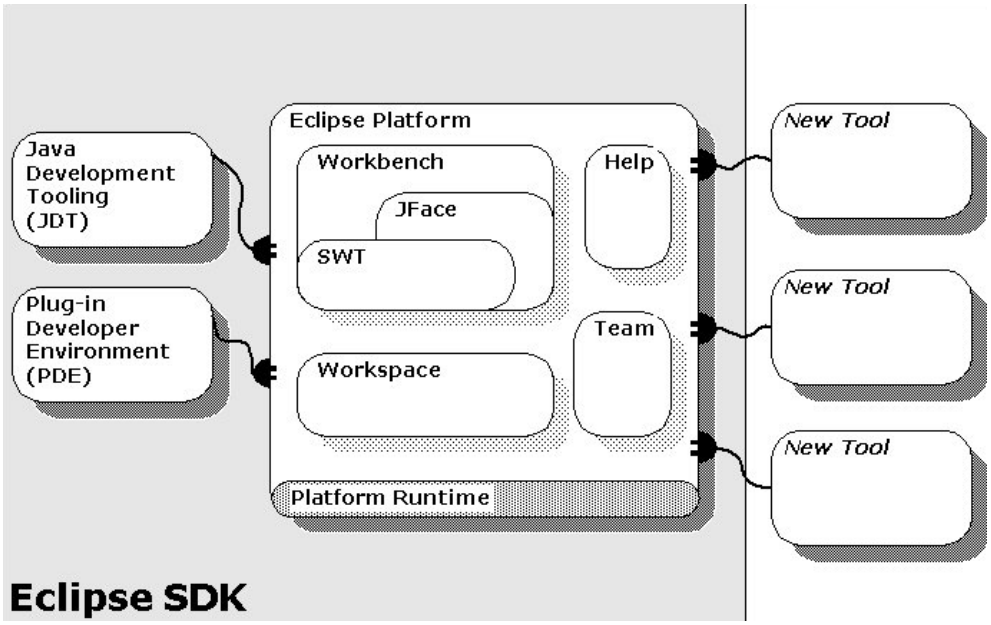
Tímto způsobem jsou od sebe implementace jednotlivých zásuvných modulů odděleny a původní modul definující body rozšíření není v době svého vývoje závislý na konkrétní implementaci modulů poskytujících jeho rozšíření [5].

## 1.2 Eclipse SDK

Na obr. 1.2 je graficky znázorněna architektura platformy Eclipse. Eclipse SDK (*Software Development Kit*) se skládá z platformy Eclipse (*Eclipse Platform*, více v sekci 1.2.1) a dvou hlavních nástrojů pro vývoj, které jsou implementovány taktéž jako zásuvné moduly [30].

**Java Development Tools (JDT)** Zásuvný modul, který poskytuje nástroje pro úpravu, zobrazení, překlad, ladění a spouštění zdrojových kódů v jazyce Java.

**Plugin-in Developer Environment (PDE)** Zásuvný modul pro vývoj, manipulaci, ladění a nasazování zásuvných modulů.




Obrázek 1.2: Struktura platformy Eclipse [30]

### 1.2.1 Platforma Eclipse

Platforma Eclipse je hlavní částí Eclipse SDK. Skládá se z několika podsystémů, které jsou implementovány jako jeden či více zásuvných modulů [5, 30].

**Jádro (*Platform runtime core*)** Definuje model bodů rozšíření a zásuvných modulů. Dynamicky vyhledává dostupné zásuvné moduly, ty jsou spouštěny až v případě potřeby po interakci s uživatelem (tzv. *lazy loading*), tímto postupem dochází k šetření paměti a výkonu. Jádro je implementováno za použití modelu OSGi (dynamický modulární systém pro programovací jazyk Java [23]).

**Správa zdrojů (*Resource management*)** Definuje API pro vytváření a správu zdrojů (projekty, soubory a složky) v souborovém systému počítače.

**Uživatelské rozhraní (*Workbench UI*)** Implementuje ovládací prvky platformy, definuje body rozšíření pro přidání prvků UI jako jsou pohledy (*view*), menu, dialogy, průvodci (*wizard*) a editory. Zajišťuje jednotný vzhled UI prvků všech zásuvných modulů.

Součástí je nízkoúrovňová knihovna grafických uživatelských prvků SWT (*Standard Widget Toolkit*), která využívá nativní knihovny OS, a knihovna JFace, která poskytuje složitější grafické prvky jako dialogy, nastavení nebo průvodce.

**Systém nápovědy (*Help system*)** Definuje body rozšíření pro nápovědu či dokumentaci, kterou lze pak procházet ve vestavěném prohlížeči. Je možné vyhledávat pomocí klíčových slov nebo dodat kontextovou nápovědu k prvkům vyvíjeného zásuvného modulu.

**Podpora spolupráce (*Team support*)** Definuje model pro verzovací systémy, umožňuje implementaci zásuvných modulů pro týmovou práci, přístup k repozitářům a verzování souborů.

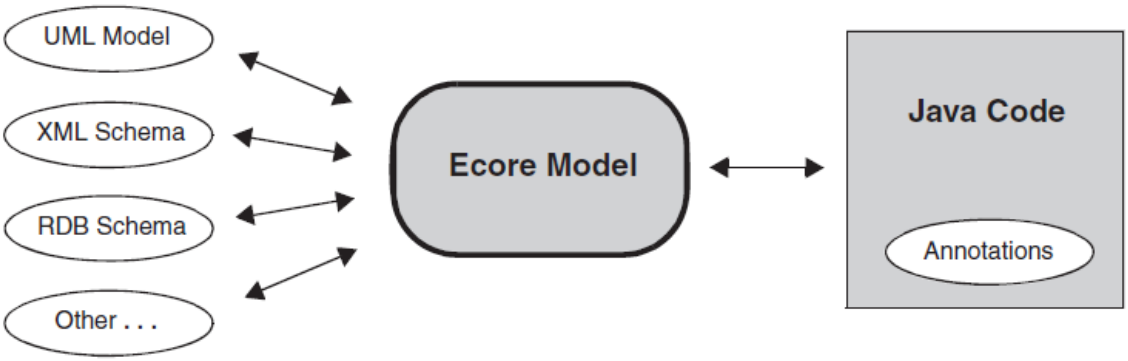
**Podpora ladění (*Debug support*)** Definuje model pro vývoj nástrojů pro ladění a spouštění procesů.

**Ostatní nástroje (*Other utilities*)** Ostatní nástroje plní funkce jako vyhledávání a porovnávání zdrojů či konfiguraci kompilace pomocí XML souborů.

## 1.3 Eclipse Modeling Framework (EMF)

Rámec EMF slouží k vývoji aplikací založených na strukturovaných datových modelech. Poskytuje UI pro tvorbu prohlížečů a editorů modelu včetně validace zadávaných dat a také umožňuje vytvářet generátory kódu. Model je vnitřně reprezentován pomocí tzv. Ecore modelu (metamodelu), jehož podoba vychází z UML [20].

Jak naznačuje obr. 1.3, informace je možné do modelu načíst z různých zdrojů, například XML Schema nebo UML model, případně také z kódu v jazyce Java opatřeném anotacemi (tento poslední přístup je zvolen v projektu VLAM IDE).




Obrázek 1.3: Zdroje Ecore modelu [20]

## 1.4 Graphical Editing Framework (GEF)

Rámec GEF umožňuje vytváření grafických editorů v Eclipse. Je implementován jako architektura MVC (*Model-View-Controller*) [5], kdy model uchovává perzistentní stavovou informaci, pohled (*View*) vykresluje obraz a zajišťuje základní interakci s uživatelem a řadič (*Controller*) koordinuje spolupráci a předávání informací mezi modelem a pohledem.

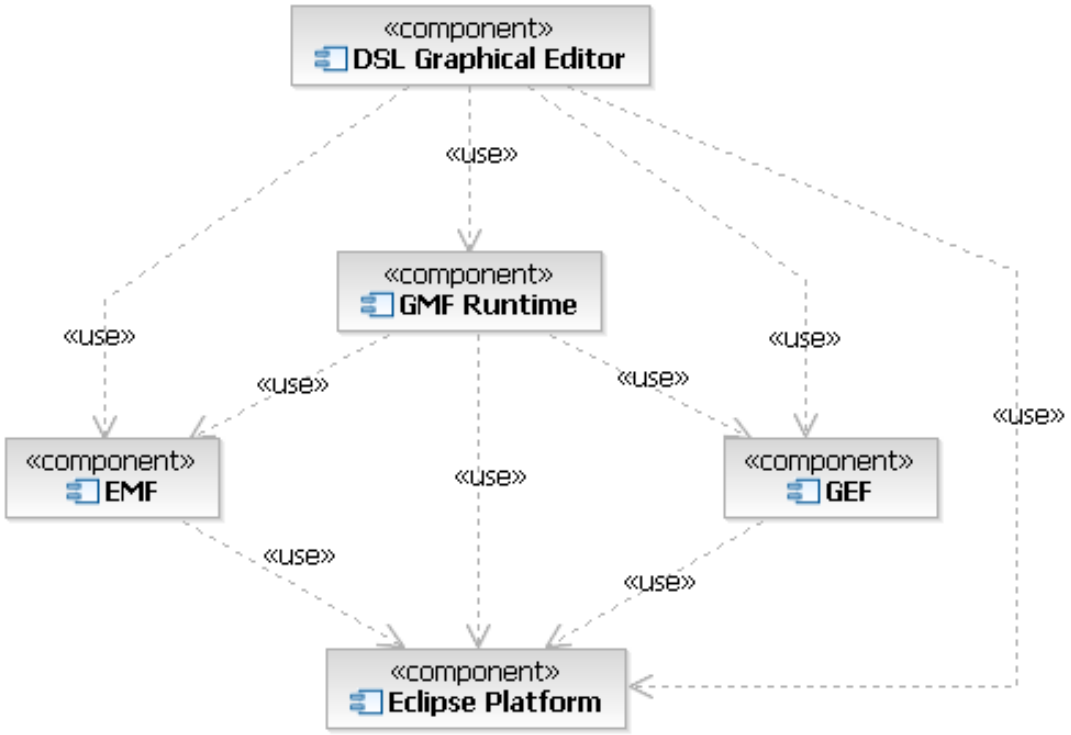
GEF rámec se skládá ze tří podčástí [5]:

- *Draw2D* – rámec nad SWT pro vykreslování grafických entit.
- *GEF Framework* – MVC rámec nad Draw2D usnadňující uživatelskou interakci v grafickém prostředí.
- *Zest* – rámec nad Draw2D pro vytváření grafů.

## 1.5 Graphical Modeling Framework (GMF)

Rámec GMF je určen k vývoji grafických editorů modelů založených na rámcích EMF (viz podkapitola 1.3) a GEF (viz podkapitola 1.4). Kombinací EMF a GEF získáváme nástroj pro tvorbu editorů určených k modelování, příkladem mohou být editory diagramů UML, modelů podnikových procesů (BPM, *Business Process Model*), diagramů datových toků (DFD, *Data Flow Diagram*) a podobně [8].

Vzájemné závislosti mezi platformou Eclipse a rámci EMF, GEF a GMF jsou znázorněny na obrázku 1.4.




Obrázek 1.4: UML diagram závislostí generovaného grafického editoru, GMF, EMF, GEF a platformy Eclipse [8]

# Kapitola 2

# Návrh vestavěných systémů

Tato kapitola se bude věnovat návrhu vestavěných systémů. Dojde tak k pokrytí teoretických základů potřebných pro návrh vyvíjeného inferenčního algoritmu, který má sloužit k hledání optimálních propojení hardwarových komponent ve schématu FPGA. Nejprve v podkapitole 2.1 vysvětlíme pojem vestavěný systém, uvedeme jeho typická a aktuální použití v praxi a čtyři nejčastější způsoby jeho implementace (FPGA a další), včetně detailnějšího popisu jejich vlastností, výhod a nevýhod. Další část kapitoly (podkapitola 2.2) se bude už zabývat metodami návrhu vestavěných systémů, zejména pak těmi, které jsou uplatňovány ve výuce.

V závěru (podkapitola 2.3) proběhne seznámení s komponentním modelováním jako s jedním z možných přístupů k návrhu vestavěných systémů a bude popsán jazyk Wright, jako jeden ze zástupců jazyků pro modelování komponentových systémů. Detailněji bude zmíněn zejména jeho strukturální popis. Tento formální popis struktury systému je využíván v komponentním editoru návrhů FPGA implementovaném v integrovaném vývojovém prostředí VLAM IDE, pro který je inferenční algoritmus navrhován. Více o prostředí VLAM IDE bude uvedeno v následující kapitole 3.

## 2.1 Vestavěný systém

Vestavěné systémy (*embedded systems*) jsou jednoúčelové počítačové systémy, můžeme je nalézt v elektrických spotřebičích, či řídicích systémech. Na rozdíl od osobních počítačů jsou méně výkonné a mají menší kapacitu paměti, jsou ovšem levnější. Mohou na nich běžet i zjednodušené operační systémy (OS), častější jsou však vestavěné systémy nevybavené OS [16]. Uveďme nyní v následující sekci pouze stručně typické aplikace vestavěných systémů.

### 2.1.1 Aplikace vestavěných systémů

Vestavěné systémy nacházíme v mnoha oblastech, v [16], přehledu zabývajícím se návrhem vestavěných systémů a jejich aplikací, jsou uvedeny ty, které jsou aktuální zejména v poslední době (rok 2011). Nacházíme zde využití v automatizaci, řídicích systémech, zpracování obrazu a zvuku, a dokonce webových službách. Vestavěné systémy nacházejí uplatnění také v komunikačních, diagnostických a robotických systémech. Dále se [16] zabývá posledními výzkumy na poli výukových platforem, ze kterých budeme čerpat v sekci 2.2.1 zabývající se metodami návrhu uplatňovanými ve výuce.

### 2.1.2 Implementace vestavěných systémů

Pro implementaci vestavěných systémů se nejčastěji používají čtyři hlavní platformy. Jsou to mikrokontroléry (MCU) a mikroprocesory, programovatelná hradlová pole (FPGA), digitální signálové procesory (DSP) a aplikačně specifické integrované obvody (ASIC). Každá implementace má své specifické vlastnosti, výhody a nevýhody, které nyní popíšeme [16].

#### Mikrokontroléry (MCU) a mikroprocesory

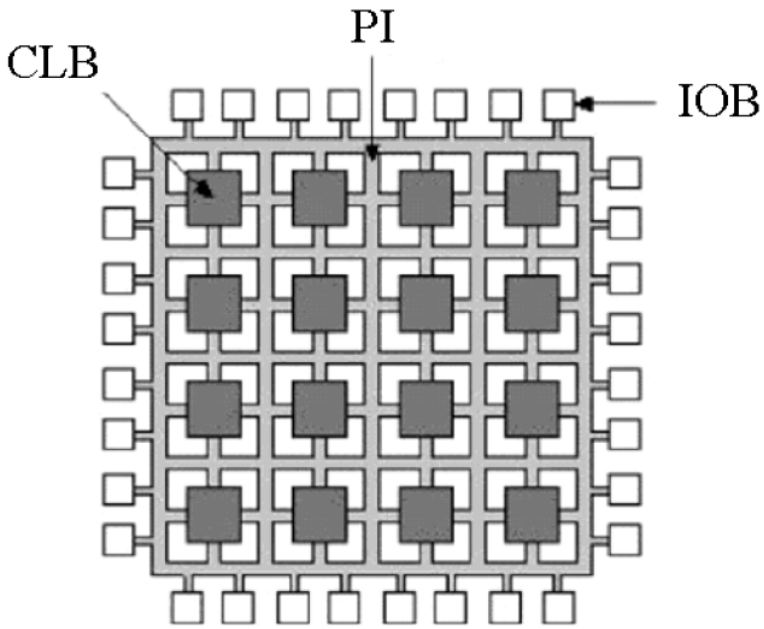
Hardwarová architektura mikroprocesorů je neměnná, mohou vykonávat sekvence základních instrukcí, které jsou typicky uloženy v paměti ROM (*read-only memory*), nebo poslední dobou většinou v paměti FLASH.

Mikrokontroléry (MCU) jsou typicky dodávány s pamětí, periferiemi a jádrem procesoru integrovanými na jediném čipu. Většinou je integrována paměť ROM, menší RAM, časovače, vstupně-výstupní obvody a sériové komunikační rozhraní.

Mikroprocesory byly vyvinuty jako jednočipová implementace CPU (*Central Processing Unit*). Dnes se můžeme setkat na jednom čipu i s více procesory, pak se jedná o tzv. vícejádrové (*multicore*) čipy. Mikroprocesory bývají vybaveny vyrovnávací pamětí RAM a hardwarovou podporou adresace virtuální paměti.

#### Programovatelná hradlová pole (FPGA)

Programovatelná hradlová pole (FPGA, *Field-Programmable Gate Arrays*) se skládají z programovatelných bloků (CLB, *Configurable Logic Blocks*) a vstupně-výstupních obvodů (IOB, *I/O Blocks*), tyto bloky mohou být vzájemně propojeny propojovací maticí (PI, *Programmable Interconnect*), viz obr. 2.1 [29].




Obrázek 2.1: Architektura FPGA (Xilinx) zahrnuje programovatelné bloky (CLB), propojovací matici (PI) a vstupně-výstupní obvody (IOB) [29].

Výhodami FPGA je rychlost provádění operací a rekonfigurovatelnost. V návrhu vestavěných systémů bývají používány buď přímo k implementaci požadované funkcionality, nebo k implementaci mikrokontrolérů nebo velmi malých procesorů (např. PicoBlaze) a potřebných periferií, což je v poslední době velmi populární přístup. Některá FPGA mohou být specializovaná na DPS (viz dále), pak obsahují na čipu i DSP bloky.

#### Digitální signálové procesory (DSP)

Digitální signálové procesory (DSP, *Digital Signal Processors*) přinášejí dostupnost komplexních aritmetických operací, je možné provádět násobení a sčítání v jedné instrukci. Jsou vhodné zejména pro zpracování číslicových signálů (obraz a zvuk), ovšem za cenu větších nákladů oproti FPGA.

#### Aplikačně specifické integrované obvody (ASIC)

Aplikačně specifické integrované obvody (ASIC, *Application-Specific Integrated Circuits*) se vyznačují vysokým výkonem, nízkou spotřebou a také nižší cenou při výrobě více kusů. V případě návrhu těchto obvodů se používá pro rychlé prototypování FPGA a pro následnou výrobu již ASIC. Ten je oproti FPGA výhodnější v případě větší produkce, protože v počátku vývoje je zapotřebí větších investic.

## 2.2 Metody návrhu vestavěných systémů

Cílem této podkapitoly je uvést přehled metod používaných při návrhu vestavěných systémů. Pro aktuálnost a uvedení zejména těch metod, které jsou dnes skutečně používány v praxi, bude tato podkapitola vycházet zejména z informací uvedených v článku [16]. Tento článek popisuje výstup rešerše provedené na aktuálních publikacích (k roku 2011) z oblasti návrhu vestavěných systémů. Zde užitečná je pak zejména část popisující přímo metody aplikované ve výuce, kterým je v této podkapitole věnována hlavní pozornost.

V literatuře se dále objevují i studie metod návrhu systémů, jejichž cílem je např. snížit náklady na výrobu nebo spotřebu energie při provozu [11]. Další se specializují na metody návrhu systémů pracujících v reálném čase [15], případně na návrh FPGA a souběžný návrh hardwarového a softwarového vybavení (více o *HW/SW co-design* viz sekce 2.2.4) [29].

Jen jako dále nerozvíjenou poznámku uveďme metodu návrhu modelem řízené architektury (MDA, *Model Driven Architecture*) pro vestavěné systémy z [15]. Tento přístup byl specifikován konzorciem OMG (*Object Management Group*) a má umožnit vývojářům vytvářet systémy za použití modelovacích technik.

### 2.2.1 Metody uplatňované ve výuce

Zaměřme se nyní již na stěžejní téma této podkapitoly, a to na metody uplatňované ve výuce (podle [16]). Začneme stručným výčtem používaných metod, které budou dále více popsány v samostatných sekcích.

Lze vysledovat, že v oblasti výuky byly pro návrh vestavěných systémů dříve používány a vyučovány především jazyky pro popis hardware (HDL, *Hardware Description Language*, více viz sekce 2.2.2). Ty byly poslední dobou nahrazeny jazyky pro popis architektur (ADL, *Architecture Description Language*, více viz sekce 2.2.3) [19].

Mimo jazyky, které již slouží k popisu výsledného hardwarového systému, je potřeba se ve výuce zabývat i prostředky pro popis požadavků na systém. To jsou podle [3] StateCharts

(rozšíření konečných automatů) a Petriho sítě, pro modelování procesů lze použít Kahnovu síť procesů (KPN, *Kahn Process Network*).

Poznamenejme, že na Fakultě informačních technologií VUT v Brně jsou studenti seznamování veskrze se všemi výše uvedenými metodami v rámci povinného předmětu zaměřeného na souběžný návrh HW a SW (Hardware/Software Codesign, viz [7]). Více o metodě souběžného návrhu programového a technického vybavení bude uvedeno v sekci 2.2.4.

### 2.2.2 HDL

Zkratkou HDL jsou označovány jazyky pro formální popis hardware (*Hardware Description Languages*). Popis může být založen na specifikaci struktury nebo chování. Mezi zástupce HDL jazyků řadíme např. jazyk VHDL nebo Verilog.

Návrh vestavěných systémů při použití HDL jazyků začíná popisem architektury, potom pokračuje vývojem instrukční sady a architektury. Následuje popis procesoru v HDL kódu a vytvoření vývojových nástrojů (např. simulátor, překladač jazyka C, asembler atd.) specifických pro navrženou architekturu.

Výhodou tohoto přístupu je extrémně malá velikost navrženého systému, nevýhodou potom je, že každá změna hardwaru znamená následné změny ve vytvořených vývojových nástrojích [19].

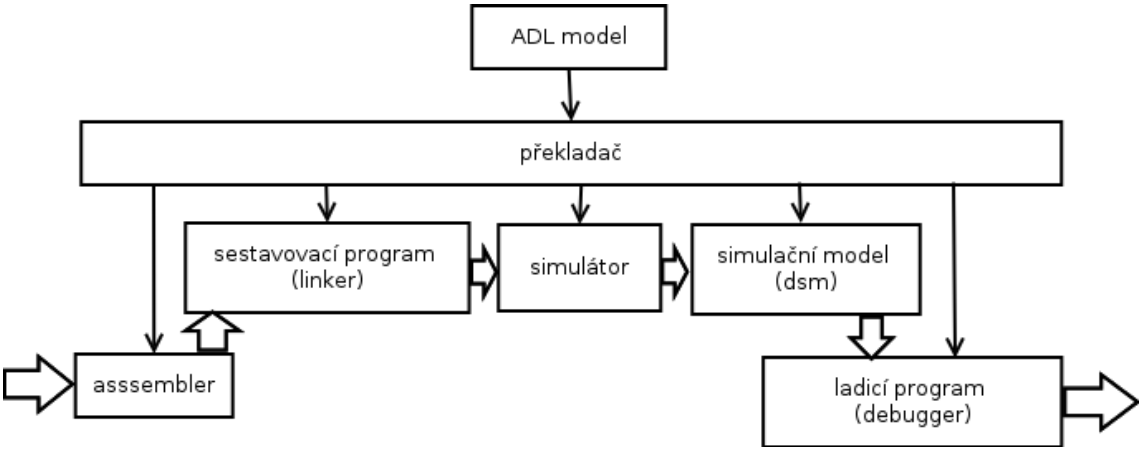
### 2.2.3 ADL

Jazyky pro popis architektury (ADL, *Architectural Description Languages*) jsou užívány pro návrh hardwarových, ale také softwarových architektur. Slouží pro vytváření aplikačně specifických procesorů (ASIP, *Application Specific Instruction-Set Processor*) nebo celých architektur pro vestavěné systémy. Z hlediska struktury popisují systém jako sadu komponent a konektorů, které zprostředkovávají vazbu mezi komponentami (více o komponentním modelování viz podkapitola 2.3).

S nástupem ADL se proces návrhu vestavěných systémů zefektivnil a zvýšila se také jeho spolehlivost. Je možné automatizovat tvorbu již výše zmíněných vývojových nástrojů (znázorněno na obr. 2.2), které byly jinak vytvářeny ručně, což s sebou přinášelo neefektivitu a chybovost. Obdobně jako u jazyků HDL, i první jazyky ADL byly rozděleny na jazyky pro popis struktury (MIMOLA, UDL/I) a chování (Valen-C, ISDL), později se začaly oba přístupy kombinovat v tzv. smíšených jazycích (nML, LISA, ICAS, TDL, EXPRESSION atd.) [19]. Mezi jazyky pro popis architektury bývá řazen i jazyk Wright (viz sekce 2.3.2), který bude popsán v další části této kapitoly zabývající se komponentním modelováním (podkapitola 2.3).

### 2.2.4 Souběžný návrh HW/SW

Architektura vestavěného systému se typicky skládá z hardware, software (ten běží na procesoru nebo ASIC) a periferních částí. Metoda souběžného návrhu technického a programového vybavení (*HW/SW co-design*) vestavěného systému sestává z několika kroků (viz obr. 2.3). Prvním je specifikace systému (popis jeho funkčnosti), další fází je rozdělení funkčnosti na hardwarovou a softwarovou část a jejich oddělená syntéza a syntéza HW/SW rozhraní. Následuje integrace systému a dále důležitá fáze souběžné simulace a verifikace (ověření, že implementace systému splňuje požadavky) vytvořeného systému [11, 29].




Obrázek 2.2: Softwarové nástroje pro vývoj mikroprocesorů při použití jazyka ADL [17].

## 2.3 Komponentní modelování

V závěru této kapitoly se podrobněji zaměříme na komponentní modelování (viz sekce 2.3.1) a speciálně jazyk Wright (viz sekce 2.3.2), jako jeden ze zástupců jazyků pro popis architektury. Jak již bylo uvedeno v úvodu této kapitoly, komponentní model, který byl využitý v editoru návrhu vestavěných systémů v integrovaném prostředí VLAM IDE, vychází právě ze strukturální části jazyka Wright.

### 2.3.1 Základní pojmy komponentního modelování

Komponentně orientovaný vývoj (CBD, *Component Based Development*) je metodologií vývoje software, která přináší velkou míru znovupoužitelnosti a rozložitelnosti. Základními pojmy této metodologie jsou *komponenta*, *rozhraní*, *konektor* a *konfigurace*. Tyto prvky tvoří dohromady tzv. *komponentový systém* (CBS, *Component Based System*) [27]. Popišme nyní jednotlivé pojmy o něco detailněji.

**Komponenta** Samostatná funkční jednotka s definovaným rozhraním, může být primitivní nebo složená, což umožňuje vystavění hierarchie komponent.

**Rozhraní** Je přístupovým bodem komponenty, je jím kompletně popsána funkcionalita komponenty, přičemž skrývá její vnitřní implementaci (tzv. *black box*).

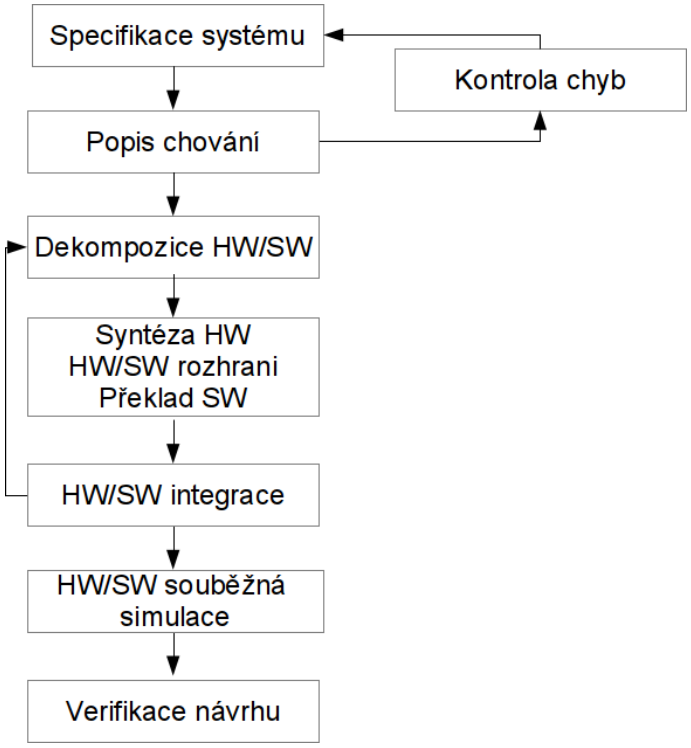
**Konektor** Zajišťuje komunikaci mezi komponentami, slouží k provázání kompatibilních rozhraní spolupracujících komponent.

**Konfigurace** Popis konkrétního propojení komponent a konektorů tvořících systém.

*Komponentový model* pak definuje syntaxi, sémantiku a kompozici komponent.

### 2.3.2 Jazyk Wright

Jazyk Wright patří mezi jazyky pro specifikaci architektur (ASL, *Architectural Specification Language*). Je vystavěn na základních pojmech známých z CBD (viz sekce 2.3.1), poskytuje formální bázi pro popis chování (behaviorální popis) a struktury (strukturální popis) architektury. Tato část vychází z [25], česká terminologie je částečně převzata z [26].




Obrázek 2.3: Souběžný návrh HW/SW: fáze vývoje [29]

### Behaviorální popis

Notace jazyka Wright vychází z formálního jazyka CSP (*Communicating Sequential Processes*), který je určen k popisu interakce konkurentních systémů. Jeho bližší popis lze najít v [27].

### Strukturální popis

Zde se zaměříme na strukturální popis, který je používán v editoru návrhu schémat. Architektury je možné v jazyce Wright popsat jako hierarchické grafy komponent a konektorů.

**Komponenta** Komponenta je definována popisem svého typu. Tento popis má dvě části, *rozhraní* (*interface*) a *sémantiku* (*computation*). Definice rozhraní komponenty se skládá z *portů*. Port popisuje dva aspekty rozhraní komponenty; částečně popisuje chování komponenty vzhledem k tomuto portu a pak očekávané chování prostředí komponenty (systému). Od základní definice CBD se tento model liší tím, že má několik portů, tedy několik rozhraní. Sémantika již na rozdíl od portů plně specifikuje chování komponenty, to je zapsáno v CSP.

**Konektory** Konektory definují vzory interakce mezi komponentami, je možné specifikovat typy a pak opakovaně vytvářet instance těchto typů. Konektor se skládá ze sady rolí (*role*) a spojovacích procesů (*glue*). *Role* popisují očekávané chování komponent, které se budou účastnit interakce. *Spojovací procesy* zajišťují koordinaci rolí (účastníků interakce), popisují tak, obdobně jako sémantika v případě komponent, již plnou specifikaci chování.

**Konfigurace** Vzniká asociací portů instancí komponent s rolemi instancí konektorů, definuje topologii a interakci komponent.

Jazyk Wright umožňuje hierarchický popis systému, sémantika portů nebo spojovací procesy konektorů mohou být popsány popisem chování v CSP, nebo popisem architektury (podsystémem).

# Kapitola 3

# Virtuální laboratoř aplikace mikroprocesorové techniky

Cílem této kapitoly bude zasazení navrhovaného inferenčního algoritmu do kontextu projektu VLAM řešeného na fakultě FIT, jeho stručnému popisu bude věnována podkapitola 3.1. Dále se detailněji zaměříme zejména na ty části projektu, jejichž podrobnější přiblížení je účelné pro následný návrh inferenčního algoritmu a uživatelského rozhraní pro jeho použití, a to na užívanou vývojovou desku FITkit (viz podkapitola 3.2) a vývojové prostředí VLAM IDE (viz podkapitola 3.3), jeden ze softwarových nástrojů, které byly v rámci řešení projektu vyvíjeny. Právě zde bude implementovaný inferenční algoritmus, jako jeden z nástrojů pro podporu návrhu vestavěného systému, ve výsledku nasazen.

## 3.1 Projekt VLAM

Projekt nese název „Virtuální laboratoř aplikace mikroprocesorové techniky“ (VLAM, *Virtual Laboratory of Microprocessor Technology Application*) a byl řešen v letech 2006-2011 pod číslem MŠMT 2C06008 [32]. Motivací projektu VLAM bylo umožnit studentům práci s laboratorním zařízením i v době, kdy není přístup do laboratoře možný (například o víkendech a svátcích), a to pomocí vzdáleného přístupu přes Internet.

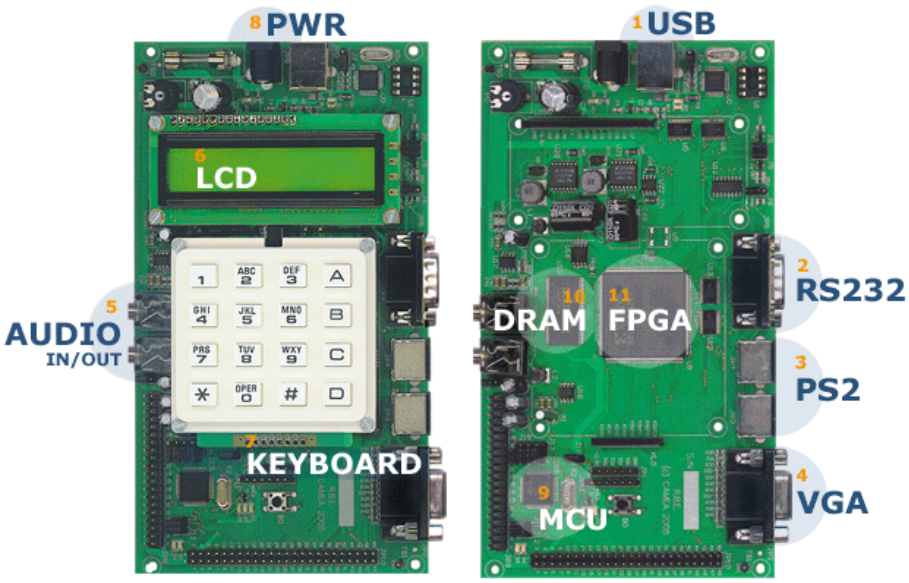
V rámci řešení vznikaly nástroje pro výuku a vývoj aplikací s mikrokontroléry a programovatelnými hradlovými poli. Součástí je podpora vzdálené práce na výukových pracovištích, která byla realizována pomocí virtuálních strojů na serveru. Byly navrženy a implementovány také nové hardwarové a softwarové nástroje pro podporu vývoje, jako VLAM IDE pro Eclipse (více v podkapitole 3.3), překladač jazyka C pro PicoBlaze a další [6]. Více detailních informací o projektu VLAM lze v případě zájmu nalézt na WWW stránkách [32].

## 3.2 Platforma FITkit

Hardwarová platforma FITkit (viz obr 3.1) je určena pro studenty fakulty FIT, kteří ji užívají ve výuce při návrhu vestavěných systémů. Přípravek obsahuje výkonný mikrokontrolér s nízkým příkonem, hradlové pole FPGA a řadu periferií. Díky poli FPGA (viz sekce 2.1.2) lze FITkit neomezeně rekonfigurovat, a vytvářet tak různé aplikace. Napájení desky a její programování probíhá přes rozhraní USB.

K vývoji aplikací pro FITkit je obvykle užívána multiplatformní aplikace QDevKit, je ovšem možné s ním pracovat i z prostředí VLAM IDE (viz podkapitola 3.3). Pro více detailů

a specifikaci platformy lze navštívit WWW stránky věnované platformě FITkit [33].




Obrázek 3.1: Přípravek FITkit při pohledu ze přední (vlevo) a zadní (vpravo) strany [33].

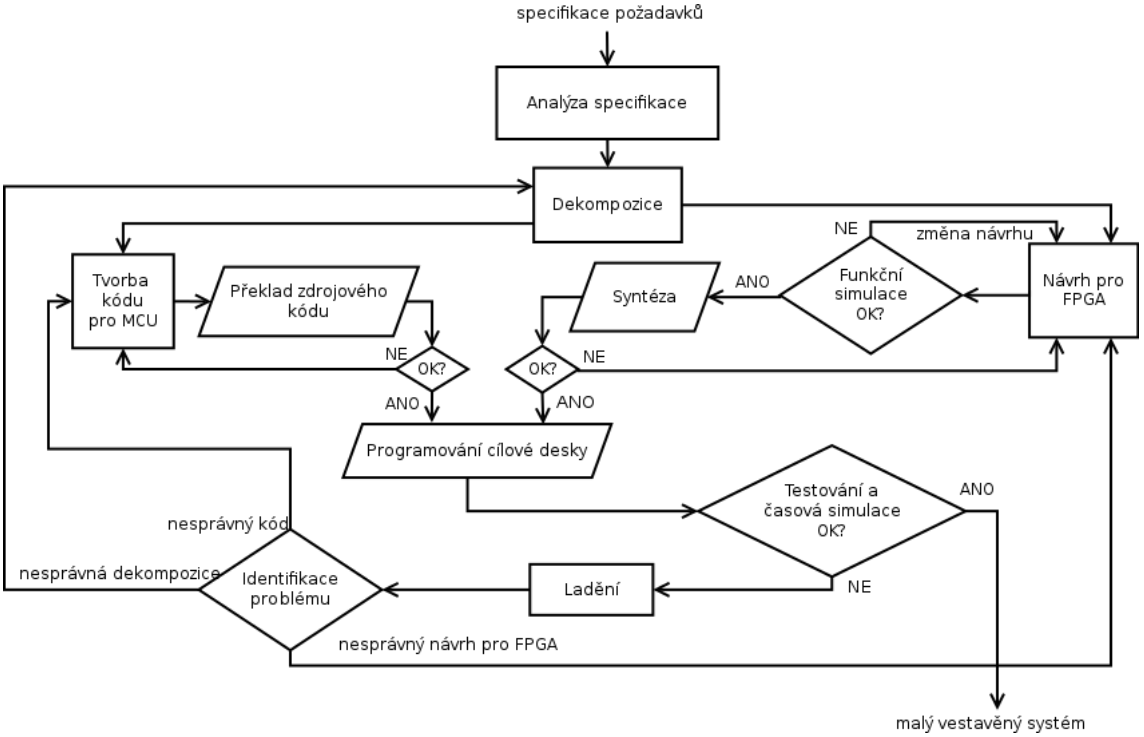
## 3.3 VLAM IDE

Vývojové prostředí VLAM IDE v sobě zahrnuje obvyklé softwarové nástroje používané při návrhu vestavěných systémů, cílovou skupinou jsou zejména začínající návrháři (studenti). Typický pracovní postup studentů, vycházející z přístupu souběžného návrhu technického a programového vybavení (*HW/SW codesign*, více viz sekce 2.2.4), znázorňuje vývojový diagram na obr. 3.2. Využívanými prostředky při vývoji, které byly do VLAM IDE začleněny, jsou potom nástroje pro

1. návrh HW pro FPGA,
2. tvorbu kódu pro procesor (MCU),
3. programování cílové desky
4. a v plánu jsou také nástroje pro simulaci, ladění a verifikaci výsledného vestavěného systému.

VLAM IDE bylo vyvíjeno jako zásuvný modul pro Eclipse (více o zásuvných modulech viz podkapitola 1.1), kdy je jednou z výhod multiplatformnost výsledného vývojového prostředí (IDE). Jeho součástí je také grafický editor pro komponentně orientovaný návrh hardwarové části pro FPGA (více o komponentním modelování viz podkapitola 2.3). Zadní část (*back-end*) editoru byla implementována nad Eclipse Modeling Framework (EMF, viz podkapitola 1.3) a přední část (*front-end*) nad Graphical Editing Framework (GEF, viz podkapitola 1.4).

Součástí IDE je tzv. báze znalostí (*knowledge base*), ta obsahuje popis hardwarových komponent, který je následně možno využít k ověření syntaktické a částečně sémantické




Obrázek 3.2: Vývojový diagram znázorňující postup studenta při návrhu vestavěného systému. Převzato a přeloženo z [10].

správnosti vytvářených propojení. Jedním z nástrojů, který tuto bázi znalostí využívá, je inferenční algoritmus vyvíjený v rámci této diplomové práce.

Poté, co uživatel navrhne hardwarové schéma v editoru, je dalším krokem vygenerování odpovídajícího kódu v jazyce pro popis hardware (např. VHDL), který popisuje vytvořená propojení. Tento kód je možné po jeho automatickém vygenerování dále upravovat v editoru, který je do IDE taktéž začleněn (Eclipse Verilog Editor, který mimo jazyka Verilog podporuje i syntaxi VHDL). Dalším krokem je pak vygenerování konfiguračního řetězce (*bitstream*) pro FPGA. Ve smyslu HW/SW souběžného návrhu je zároveň možné vytvářet v editoru kód pro procesor (v jazycích asembler a C) a následně jej přeložit odpovídajícím překladačem. Z VLAM IDE jsou volány všechny potřebné externí nástroje s příkazovým rozhraním, jako překladače, asemblery, simulátory a ladicí programy. Jejich automatické nastavení zajišťuje konfigurace uživatelského projektu v Eclipse. V následující sekci 3.3.1 se zaměříme na bližší popis uživatelského rozhraní VLAM IDE.

### 3.3.1 Uživatelské rozhraní

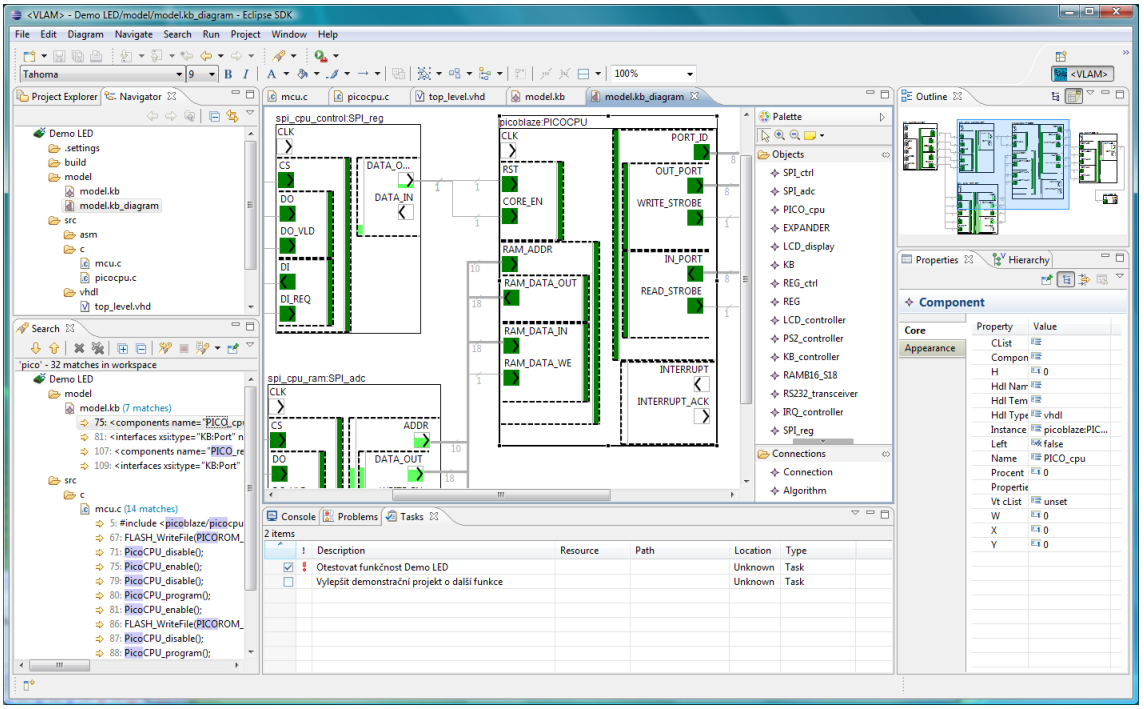
Uživatelské rozhraní ve standardní podobě je znázorněno na obr. 3.3. Skládá se z několika speciálních panelů (*Views*) [6]:

1. *Komponentní editor návrhu (uprostřed nahoře)* Skládá se z návrhové plochy hardwarového diagramu a palety nástrojů obsahující komponenty a propojení (více viz sekce 3.3.2).

2. *Náhled na diagram (vpravo nahoře)* Zobrazuje přehledně celý diagram.

3. *Vlastnosti (vpravo dole)* Pro zobrazení vlastností vybrané entity.

4. *Navigátor (vlevo nahoře)* Je určený pro zobrazení zdrojů v uživatelském projektu, jako jsou modely a diagramy, soubory se zdrojovým kódem, vygenerované konfigurační řetězce a binární soubory a logovací soubory.




Obrázek 3.3: Ukázka vývojového prostředí VLAM IDE [6].

### 3.3.2 Komponentní editor návrhu

Editor návrhu (na obr. 3.3 uprostřed nahoře) je implementován jako zásuvný modul pro Eclipse a ve vyvíjeném IDE má dvě úlohy. Slouží jako grafický modelovací nástroj pro tvorbu hardwarových diagramů a druhou úlohou je poskytování uživatelského rozhraní pro inferenční algoritmus.

Jak již bylo zmíněno v úvodu, implementace komponentního editoru je založena na rámcích Eclipse Modeling Framework (pro definici modelu) a Graphical Editing Framework (pro zobrazení modelu), který využívá Graphical Modeling Framework pro vygenerování grafického editoru modelu.

Na návrhovém plátně, které je součástí editoru, jsou zobrazovány komponenty a jejich propojení. Každá komponenta má definováno rozhraní, které tvoří hierarchická struktura portů a pinů. Při vytváření návrhu je používána paleta (*Palette*), ze které je možné komponenty vybírat, kliknutím umístit na plátno a následně je propojovat buď manuálně (použitím nástroje *Connection*), nebo polautomaticky pomocí inferenčního algoritmu (nástroj *Algorithm*). V editoru je dále možné komponenty a propojení z návrhu i odebírat, případně měnit jejich vlastnosti.

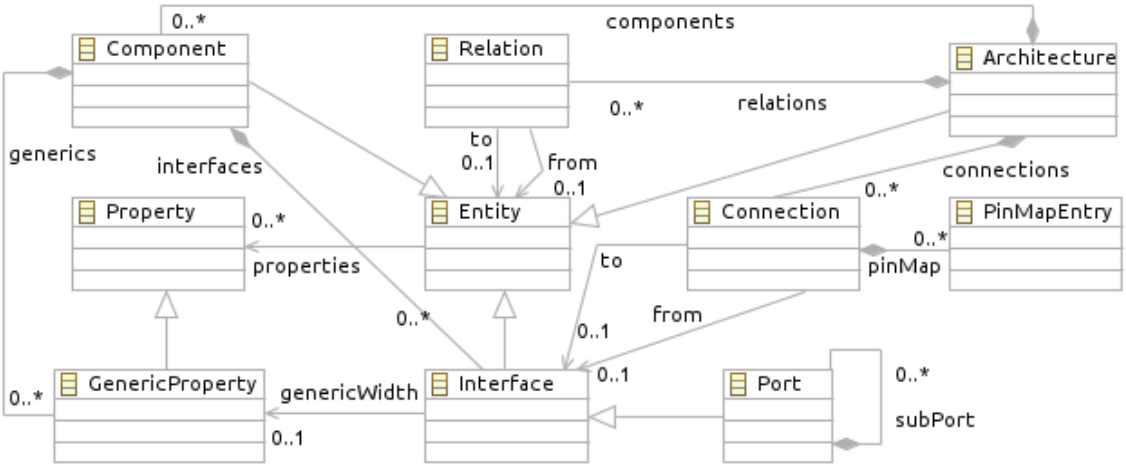
### 3.3.3 Model popisu hardwarového schématu

Model popisu hardwarového schématu v prostředí VLAM IDE popíšeme návrhovým diagramem tříd pro EMF znázorněném na obr. 3.4. Konfigurace je modelována pomocí entit (*Entity*), což mohou být komponenty (*Component*) nebo porty (*Port*). Entity jsou popsány atributy, jako jsou jméno, omezení a výchozí nastavení různých parametrů a údaje pro generování kódu. Komponenty se skládají z portů, ty mají obecně hierarchickou strukturu, kdy porty mohou obsahovat další porty a piny (port šířky jedna). Hierarchické členění umožňuje sdružovat porty do skupin podle jejich účelu.

Vypuštěním třídy *Connection* získáváme metamodel báze znalostí (*knowledge base*, KB). V KB jsou uloženy tyto vlastnosti portů:

- *Směr* Vstupní (*In*), výstupní (*Out*), vstupně-výstupní (*Inout*).
- *Typ* Datový, adresový, řídicí.
- *Šířka* Určuje kapacitu portů. Porty mohou mít v konfiguraci také již částečně obsazenou šířku, pokud se komponenta účastní ve vytvářeném schématu také v propojení s jinými komponentami.
- *Role* Určují sémantiku portů, ta je v bázi znalostí vyjádřena prioritním seznamem možných rolí portu (koncept rolí vychází ze strukturální části jazyka Wright, viz sekce 2.3.2).

Hodnota vlastnosti může být dána *generickým parametrem*[1], v tomto případě ji zadává uživatel až při instanciaci entity, případně je odvozována automaticky. Množina povolených hodnot generických parametrů je dána třídou *GenericProperty* [9].




Obrázek 3.4: Stručný UML diagram tříd modelující konfiguraci.

[1] Konstrukt v jazyce VHDL.

# Kapitola 4

# Bipartitní párování a přiřazovací problémy

V této kapitole bude jako první zmíněna problematika bipartitního párování (podkapitola 4.1), které je zobecněním přiřazovacích problémů (AP, *Assignment Problem*). Popis bude doplněn o základní principy algoritmů pro řešení různých typů párování. V následujících částech kapitoly se potom budeme věnovat již jednotlivým typům AP.

Oblast AP se zabývá hledáním přiřazení $n$ zdrojů k $n$ prostředkům, problémy tohoto typu patří do oblasti kombinatorické optimalizace. Příkladem může být přídělení úkolů agentům, plánování rozvrhu do učeben, či návrh optimálního rozmístění komponent na desce. Podle způsobu výpočtu ohodnocení přiřazení a jeho vlastností se dále rozděluje na základní podtřídy: lineární AP (LAP, *Linear AP*, viz podkapitola 4.2), kvadratický AP (QAP, *Quadratic AP*, viz podkapitola 4.4), obecný AP (GAP, *Generalized AP*, viz podkapitola 4.3), a případně i jejich kombinace.

Obsah této kapitoly je stěžejní pro následovný návrh algoritmu pro hledání nejlepšího propojení komponent. Největší pozornost bude věnována problému maximálního bipartitního párování a algoritmu jeho řešení (viz sekce 4.1.1) včetně jeho vylepšené verze (viz sekce 4.1.2), která bude základem implementace navrhovaného algoritmu.

## 4.1 Bipartitní párování

Problém bipartitního párování (BM, *Bipartite Matching*) je, jak už bylo výše zmíněno, zobecněním AP. Algoritmy vedoucí k jeho řešení patří do rodiny tzv. síťových algoritmů (*Network Algorithms*). Uveďme nyní první definice vedoucí k zavedení jednotlivých typů BM (citováno z [12]).

**Definice 4.1** Orientovaný graf $G$ *je dvojice* $G = (V, E)$, *kde* $V$ *je konečná množina* uzlů *a* $E \subseteq V^2$ *je množina* hran. *Pokud* $(u, v)$ *je hrana, říkáme, že* $u$ *a* $v$ *jsou* incidentní.

**Definice 4.2** Neorientovaný graf $G$ *je dvojice* $G = (V, E)$, *kde* $V$ *je konečná množina* uzlů *a* $E \subseteq \binom{V}{2}$ *je množina* hran.

**Definice 4.3** Bipartitní graf *je takový graf* $G = (V, E)$, *kde* $V$ *se dá rozložit na* $V = L \cup R$, $R \cap L = \emptyset$ *a* $E \subseteq L \times R$.

**Definice 4.4** Párování *v $G$ je podmnožina hran $M \subseteq E$ taková, že pro každý uzel $v \in V$, $v$ je incidentní s nejvýše jednou hranou z $M$.*

**Definice 4.5** Maximální párování *je párování s maximální kardinalitou.*

Pro řešení maximálního bipartitního párování bývá používána Ford-Fulkersonova metoda [12], detailnější rozbor včetně implementace lze nalézt také v [18]. Pro zavedení ceny párování a cenově maximálního párování potřebujeme definovat další pojmy [18].

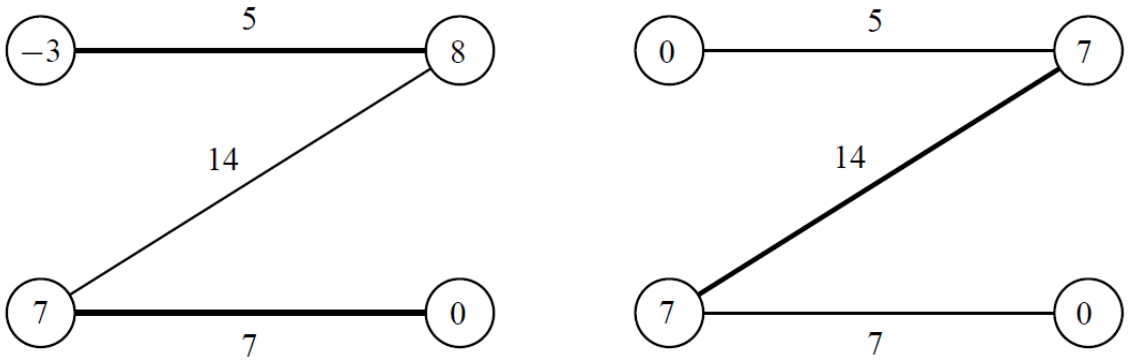
**Definice 4.6** Funkce váhy (ceny) hrany *je funkce $c : E \mapsto \mathbb{R}$.*

**Definice 4.7** Cena párování $M$ *je suma cen jeho hran, tedy* $c(M) = \sum_{e \in M} c(e)$.

**Definice 4.8** Maximální váhované párování (MWM, Maximum Weight Matching) *nazýváme párování, jehož cena je nejméně tak vysoká jako cena ostatních možných párování.*

**Definice 4.9** Maximální váhované přiřazení (Maximum Weight Assignment) *nazýváme nejvýše ohodnocené přiřazení (tedy úplné párování, při kterém je každý uzel incidentní právě s jednou hranou).*

Na obrázku 4.1 je znázorněn rozdíl mezi problémy z definic pro párování (definice 4.8) a přiřazení (definice 4.9).




Obrázek 4.1: Maximální váhované přiřazení (nalevo) a párování (napravo). Čísla v uzlech jsou pomocnými hodnotami používanými algoritmy, jedná se o tzv. *potenciál uzlu* [18].

Pro úplnost zmiňme ještě další modifikace bipartitního párování a přiřazení, minimální váhované přiřazení (definice 4.10) a maximálně váhované maximální párování (definice 4.11).

**Definice 4.10** Minimální váhované přiřazení (Minimum Weight Assignment) *nazýváme nejníže ohodnocené přiřazení.*

**Definice 4.11** Maximálně váhované maximální párování (Maximum Weight Maximal Cardinality Matching) *nazýváme nejvýše ohodnocené párování ze všech párování maximální kardinality.*

V následujících sekcích se seznámíme s algoritmy pro řešení situací z výše uvedených definic. Popisy algoritmů jsou citovány z [18].

### 4.1.1 Algoritmus pro maximální váhované párování

Pro zavedení algoritmu pro řešení MWM je potřeba blíže specifikovat další pojmy z oblasti bipartitního párování.

**Definice 4.12** *Mějme bipartitní graf* $G = (A \cup B, E)$ *a ohodnocovací funkci* $c : E \mapsto \mathbb{R}$ *hran v* $G$. Párování $M$ *je podmnožinou* $E$ *takovou, že žádné dvě hrany nesdílejí incidentní uzel. Uzel* $v$ *nazýváme* spárovaný *vzhledem k párování* $M$*, pokud existuje hrana v* $M$ *incidentní k* $v$*, jinak jde o uzel* volný *neboli* nespárovaný*.*

Algoritmus dále pracuje s tzv. potenciální funkcí (viz definice 4.13) a jejími vlastnostmi.

**Definice 4.13** Potenicální funkcí *nazýváme funkci* $\pi : V \mapsto \mathbb{R}$ *a pro libovolnou hranu* $e = (v, w)$ *nazýváme* $\overline{c}(e) = \pi(v) + \pi(w) - c(e)$ redukovanou cenou (reduced cost) *hrany* $e$ *vzhledem k* $\pi$*. Hranu nazýváme* úzkou (tight)*, pokud její redukovaná cena je nulová.*

V algoritmu jsou využívány tyto čtyři vlastnosti potenciální funkce:

1. nezáporná hodnota redukovaných cen, tj. $\overline{c}(e) \geq 0$ pro všechny $e \in E$,
2. hrany účastnící se na párování jsou úzké, tj. $\overline{c}(e) = 0$ pro $e \in M$,
3. potenciál uzlu je nezáporné číslo, tj. $\pi(v) \geq 0$ pro všechna $v \in V$,
4. volné uzly mají nulový potenciál, tj. $\pi(v) = 0$ pro všechna $v$, která jsou volná vzhledem k párování $M$.

**Definice 4.14** *Potenciální funkce je* dostačující (feasible)*, pokud splňuje vlastnost 1,* nezáporná*, pokud splňuje vlastnost 3, a* úzká*, pokud splňuje vlastnosti 1, 2 a 4.*

Na vlastnostech z definice 4.14 je založeno ověření optimálnosti nalezeného párování resp. přiřazení (více v [18]).

Algoritmus prezentovaný v [18] pracuje iterativně, začíná s prázdnou množinou hran účastnících se na párování $M$, grafem indukovaným (*spanned by*) $B$ a prázdnou podmnožinou množiny $A$. Poté jsou postupně do grafu přidávány uzly z $A$, po každé iteraci je vypočítáno nové párování a nová úzká potenciální funkce.

#### Detailní popis algoritmu

Nechť $a_1, ..., a_n$ je výčet prvků v $A$, $A_i = \{a_1, ..., a_i\}$, $G_i$ podgraf indukovaný $V_i = A_i \cup B$, $M_i$ maximální váhované párování v $G_i$ a $\pi_i : V_i \mapsto \mathbb{R}_{\geq 0}$ nezáporná potenciální funkce, která je úzká vzhledem k $M_i$. Algoritmus konstruuje postupně $M_i$ a $\pi_i$ pro $i = 0, 1, ..., n$. Pracujeme s orientovaným grafem, je dáno, že všechny hrany účastnící se na párování jsou orientovány z $B$ do $A$ a neúčastnící se z $A$ do $B$.

Nejprve vytvoříme $M_0$ a $\pi_0$; $M_0$ je prázdné párování a $\pi_0$ přiřazuje nulu pro všechny uzly v $B$. Pro získání $M_1$ a $\pi_1$ mějme $e$ nejvýše ohodnocenou hranou incidentní k $a_1$. Pokud $e$ neexistuje nebo je její cena záporná, potom $M_1$ je prázdná a $\pi_1$ přiřazuje nulový potenciál všem vrcholům ve $V_1$. Jinak je $M_1 = \{e\}$ a $\pi_1$ přiřazuje potenciál $c(e)$ uzlu $a_1$ a nulu všem uzlům v $B$.

Konstrukce $M_i$ a $\pi_i$, pokud známe $M_{i-1}$ a $\pi_{i-1}$, probíhá následovně. Rozšíříme definici $\pi_{i-1}$ na dostačující nezápornou potenciální funkci $\overline{\pi_i}$ pro $V_i$. Toto může být provedeno nastavením $\overline{\pi_i}(a_i)$ na libovolnou hodnotu, která zaručí, že redukované ceny hran incidentních k $a_i$ budou nezáporné. Dále mějme $M = M_{i-1}$, $\pi = \overline{\pi}_i$ a $a = a_i$ a upravujeme funkci $\pi$.

Úpravy funkce $\pi$ probíhají ve fázích. V každé fázi určujeme množinu $R$ uzlů, které jsou dosažitelné z $a$ přes úzké hrany, a poté rozlišujeme 3 případy:

**$R \cap A$ obsahuje uzel s nulovým potenciálem** Mějme $v$ uzel v $R \cap A$ s $\pi(v) = 0$ a $p$ cestu po úzkých hranách z $a$ do $v$. Pokud nastane tento případ, $a = v$ a $p$ je délky 0. Nastavíme $M_i = M$ a $\pi_i = \pi$.

**$R \cap B$ obsahuje volný uzel** Nechť $w$ je volným uzlem v $R \cap B$ a $p$ cesta po uzkých hranách z $a$ do $w$. Nastavíme $M_i = M \oplus p$ (augmentace párování $M$ podle $p$, tedy změny orientace hran na cestě $p$) a $\pi_i = \pi$.

**ani jedno výše uvedené** Definujeme hodnotu $\sigma = min(\alpha, \beta)$. Nechť $\alpha$ je minimální hodnota $\pi(v)$ pro libovolný uzel $v \in R \cap A$ a $\beta$ minimální hodnota redukované ceny $\overline{c}(e)$ libovolné hrany $e$ vedoucí z $R$. O hodnotu $\sigma$ snížíme potenciál všech uzlů v $R \cap A$ a zvýšíme potenciál všech uzlů v $R \cap B$, poté přepočítáme $R$. Takto pokračujeme, dokud nenastane některý z prvních dvou případů.

**Popis algoritmu v pseudokódu**

```
M = prázdné přiřazení;
π(b) = 0 pro všechna b ∈ B;

forall a ∈ A
{
  nastav π(a) na hodnotu, při které jsou redukované hodnoty všech hran
  incidentních k a nezáporné;

  while (true)
  {
    urči množinu R uzlů dosažitelných z a po úzkých hranách;
    if R obsahuje uzel v A nulového potenciálu nebo volný uzel v B
    {
      augmentuj přiřazení úzkou cestou k tomuto uzlu;
      break;
    }
    vypočítej α, β a σ a uprav potenciály;
  }
}
```

Důkaz správnosti algoritmu je uveden v [18]. V další podkapitole se zaměříme na vylepšenou verzi algoritmu, která bude nakonec v navrhovaném inferenčním algoritmu implementována.

### 4.1.2 Vylepšená verze algoritmu pro maximální váhované párování

Základní podoba algoritmu uvedená výše v sekci 4.1.1 je v [18] dále vylepšována z hlediska časové složitosti, jsou zaváděny principy hledání nejkratší cesty při konstrukci pomocných množin a heuristiky pro počáteční nastavení potenciálů uzlů. Takto upravený algoritmus má potom časovou složitost nejhoršího případu $O(n(m + n \cdot log\, n))$, kde $n$ je počet uzlů a $m$ počet hran v grafu.

Je modifikována vnitřní smyčka algoritmu, která zvětšuje množinu $R$, dokud $R$ obsahuje volný uzel v $B$ nebo uzel v $A$ s nulovým potenciálem. Mějme $R_k$ množinu $R$ v $k$-té iteraci smyčky pro $k = 1, 2, ..., K + 1$, $\delta_k$ hodnotu $\delta$ v $k$-té iteraci, $\Delta_k = \delta_1 + ... + \delta_k$ a $\Delta = \Delta_K$ sumu všech $\delta$. Potom celková změna potenciálů uzlů v $R_k \setminus R_{k-1}$ je $\delta_k + \delta_{k+1} + ... = \Delta - (\delta_1 + ... + \delta_{k-1})$. Zvětšování množiny $R$ vztahujeme k výpočtu nejkratších cest s počátkem v uzlu $a_i$.

Úprava algoritmu vychází z platnosti následujících dvou vět (důkaz v [18]).

**Věta 4.1.1** *Mějme libovolný uzel $w$ a $\mu(w)$ vzdálenost nejkratší cesty z $a_i$ do $w$ vzhledem k redukovaným cenám definovaným $\overline{\pi}_i$. Potom je $w$ přidán k $R$ poté, co dojde k celkové změně potenciálu o velikosti $\mu(w)$.*

**Věta 4.1.2** *Mějme $\pi = \overline{\pi}_i$ a $\mu(w)$ vzdálenost nejkratší cesty z $a_i$ do uzlu $w$ vzhledem k redukovaným cenám definovaným $\pi$. Mějme $minA = min\{\mu(a) + \pi(a); a \in A\}$ a $minB = min\{\mu(b) + \pi(b);\ b \in B$ a $b$ je volný$\}$. Potom $\Delta = min(minA, minB)$ a změna celkového potenciálu pro libovolný uzel $v$ je rovna $max(0, \Delta - \mu(v))$.*

*Pokud $\Delta = minA$, mějme $z$ uzel určující $minA$, a pokud $\Delta = minB$, mějme $z$ uzel určující $minB$. V obou případech mějme $p$ cestu z $a_i$ do $z$ o délce $\mu(z)$. Potom jsou všechny hrany po této změně potenciálu úzké.*

### Popis algoritmu v pseudokódu

```
M = prázdné přiřazení;
π(b) = 0 pro všechna b ∈ B;

forall a ∈ A
{
  nastav π(a) na hodnotu, při které jsou redukované hodnoty všech hran
  incidentních k a nezáporné;

  pro libovolný uzel v nechť je dist(v) vzdálenost nejkratší cesty
  z a do v;

  minA = min{dist(v) + pot(v); v ∈ A};
  nejlepsi_uzel_v_A = uzel definující minA;

  minB = min{dist(v); v ∈ B a je volný};
  nejlepsi_uzel_v_B = uzel definující minB;

  Delta = min(minA, minB);

  forall v ∈ A: pot(v) − max(0, Delta − dist(v));
  forall v ∈ B: pot(v) + max(0, Delta − dist(v));

  augmentuj přiřazení úzkou cestou z a do nejlepsi_uzel_v_A, pokud
  Delta = minA, jinak do nejlepsi_uzel_v_B;
}
```

#### Implementace algoritmu v knihovně LEDA

V této své nejefektivnější podobě bude algoritmus nakonec implementován, proto potřebné bližší implementační detaily z [18] jsou k nalezení v kapitole 6 této diplomové práce, která se věnuje implementaci.

Zmiňme pouze, že základ algoritmu je modifikací Dijsktrova algoritmu pro hledání nejkratších cest. Knihovna LEDA [18] obsahuje implementace různých grafových algoritmu v jazyce C++, úkolem bude tedy inspirovat se touto implementací párovacího algoritmu a přenést ji do jazyka Java (výslednou implementaci algoritmu v jazyce Java popisuje podkapitola 6.1).

### 4.1.3 Další algoritmy pro bipartitní párování

Úpravou výše uvedeného algoritmu dostáváme podle [18] algoritmus pro řešení maximálního (resp. minimálního) váhovaného přiřazení (definice 4.9), tedy lineárního přiřazovacího problému, o kterém bude více pojednáno v podkapitole 4.2. Úprava vynechává první případ při úpravách potenciální funkce a vždy se snaží vyhledat cestu až k volnému uzlu množiny $B$. Pokud v $B$ není nikdy nalezen volný uzel, pak graf neobsahuje žádné přiřazení.

Pro minimální váhované přiřazení (definice 4.10) je algoritmus inicializován změnou znaménka v ohodnocení hran. Obě výše zmíněné varianty mají stejnou časovou složitost nejhoršího případu jako upravený základní algoritmus, tedy $O(n(m + n \cdot log\, n))$, kde $n$ je počet uzlů a $m$ počet hran v grafu.

Pro hledání maximálně váhovaného maximálního párování (definice 4.11) je principem úpravy základního algoritmu zvýšení ohodnocení hran v grafu o reálné číslo $L$, tedy $c_L(e) = c(e) + L$ pro každé $c \in E$, tato úprava ohodnocení zřejmě favorizuje při vyšších hodnotách $L$ párování vyšší kardinality. Poté již postupujeme podle základního algoritmu pro maximálně váhované párování. Přesný postup získání hodnoty $L$ je uveden v [18].

Další text bude věnován maximálnímu váhovanému přiřazení, tedy přiřazovacímu problému (AP). Pokud nebude uvedeno jinak, další definice a popisy jsou citovány z [24], článku, který shrnuje publikovaný výzkum do roku 2005.

## 4.2 Lineární AP (LAP)

LAP je definován jako přiřazení $n$ úloh $n$ agentům, příkladem může být přiřazení pracovníků ke strojům nebo úlohám. Matematicky popíšeme LAP takto:

Hledáme minimum funkce:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

Za podmínek:

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, ..., n,$$

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., n,$$

$$x_{ij} = 0 \; or \; 1,$$

kde $x_{ij} = 1$, pokud je agent $i$ přiřazen k úloze $j$, 0 pokud není, a $c_{ij}$ je cena přiřazení agenta $i$ k úloze $j$. Nejznámější metodou řešení základního LAP je tzv. Maďarský algoritmus.

### 4.2.1 Algoritmy pro řešení LAP

Maďarský algoritmus má dlouhou historii a stál na počátku oboru kombinatorické optimalizace. Pochází z padesátých let minulého století, kdy jeho první verze měla časovou složitost $O(n^4)$. Vylepšení na $O(n^3)$ přinesla adaptace algoritmu pro hledání nejkratší cesty (Dijkstrův algoritmus) pro problém LAP (Edmond a Karp v roce 1969).

Různé implementace Maďarského algoritmu byly po dlouhou dobu nejúspěšnějším nástrojem pro řešení instancí LAP. V roce 1989 byl vylepšen Gabowem a Tarjanem na časovou složitost $O(\sqrt{n} \cdot m \cdot log(nC))$, kde $C = max_{i,j}\{c_{ij}\}$, a to za použití tzv. zjemňující techniky (*cost scaling technique*). Ve výčtu historie algoritmů v [4] jsou dále uvedena i řešení LAP paralelními algoritmy.

LAP může být řešen i variantou algoritmu pro maximální váhované bipartitní párování, jak bylo již zmíněno v sekci 4.1.3.

### 4.2.2 Modifikace LAP

LAP nemusí být vždy zadan přesně v souladu s definicí uvedenou výše, nicméně jiná zadání lze na tuto definici snadno převést. Jedná se například o hledání propojení maximální ceny, v tomto případě nahradíme hodnoty $c_{ij}$ hodnotou $c_{max} - c_{ij}$, kde $c_{max}$ je maximum hodnot $c_{ij}$.

V případě, kdy počet úloh a agentů není roven, řešíme úlohu doplněním fiktivních agentů (resp. úloh) s nulovým oceněním jejich možných přiřazení. Problém můžeme dále modifikovat případným omezením přiřazení agenta k určitému úkolu (nekompatibilita) a následně jej řešit metodami pro LAP (více v [24]).

## 4.3 Obecný AP (GAP)

Obecným přiřazovacím problémem nazýváme situaci, kdy můžeme přiřadit agentovi více úloh najednou. Matematicky lze vyjádřit zadání takto:

Hledáme minimum funkce:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$$

Za podmínek:

$$\sum_{i=1}^{m} x_{ij} = 1, j = 1, ..., n,$$

$$\sum_{j=1}^{n} a_{ij}x_{ij} \leq b_i, i = 1, ..., m,$$

$$x_{ij} = 0 \; or \; 1,$$

kde $x_{ij} = 1$, pokud je agent $i$ přiřazen k úloze $j$, 0 pokud není, a $c_{ij}$ je cena přiřazení agenta

$i$ k úloze $j$, $a_{ij}$ je velikost kapacity agenta při přiřazení k úloze $j$ a $b_i$ je dostupná kapacita agenta $i$. Tento problém je řešen např. při návrhu komunikačních sítí nebo plánování reklamního času.

Tento problém, na rozdíl od předchozího LAP, je již ve třídě NP [28], proto algoritmy, které jej řeší, přinášejí pouze aproximaci nejlepšího řešení. V literatuře nacházíme řešení pomocí heuristik [14] nebo rojové inteligence [35].

## 4.4 Kvadratický AP (QAP)

Typickou úlohou pro přiblížení významu QAP je problém přiřazení $n$ aktivit $m$ ($m \leq n$) lokalitám, ohodnocení výsledného přiřazení je přitom závislé na toku mezi aktivitami a vzdálenosti příslušných lokalit. Matematicky vyjádřeno:

Hledáme minimum funkce:

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=1}^{n} c_{ijpq}x_{ij}x_{pq}$$

Za podmínek:

$$\sum_{i=1}^{m} x_{ij} \leq 1, j = 1, ..., n,$$

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, ..., m,$$

$$x_{ij} = 0 \; or \; 1,$$

kde $x_{ij} = 1$, pokud je agent $i$ přiřazen k úloze $j$, 0 pokud není. Cena $c_{ijpq}$, kterou je ohodnocena interakce mezi aktivitou $i$ v lokalitě $j$ a aktivitou $p$ v lokalitě $q$, je určena jako součin toku mezi aktivitami $i$ a $p$ a vzdálenosti mezi lokalitami $j$ a $q$.

Jako QAP popisujeme tedy případ, kdy do hodnotící funkce vstupuje závislost mezi jednotlivými aktivitami (a/nebo lokalitami). Tento problém je opět, stejně jako GAP (viz podkapitola 4.3), ve třídě NP [28]. Pro svou častou zastoupenost v problémech reálného světa byl již mnohokrát zkoumán a pro jeho řešení bylo použito široké spektrum metod a algoritmů.

Z reálných problémů QAP modeluje například problém návrhu rozložení kláves na klávesnici (*keyboard design*), je potřeba rozmístit klávesy tak, aby psaní textů trvalo co nejkratší dobu. Jako vstup je přitom použita frekvenční analýza jazyka (frekvence dvojznaků). Obdobným případem z ergonomické praxe je návrh kontrolních panelů tak, aby byla minimalizována únava očí.

Speciálními případy QAP jsou problém obchodního cestujícího, problém nalezení největší kliky v grafu a problém známý pod názvem „rozdělení kořisti“ (*graph partitioning problem*) [4].

### 4.4.1 Algoritmy pro řešení QAP

Metodami řešení QAP jsou různé heuristiky, metody rojové inteligence, evoluční algoritmy, metoda větví a mezí (*Branch-and-Bound*) či metoda větví a řezu (*Branch-and-Cut*), dynamické programování, simulované žíhání. Přehled metod zkoumaných do roku 2005 můžeme

najít v [13]. Nejnovější články zmiňují využití metod lineárního programování, Fourierova prostoru (*Fourier Space*) nebo zakázaného prohledávání (*Tabu Search*).

# Kapitola 5

# Návrh inferenčního algoritmu

V této kapitole se zaměříme na požadované vlastnosti inferenčního algoritmu pro propojování komponent a jeho samotný návrh. Bude rozebrána úloha algoritmu v editoru návrhů pro FPGA ve VLAM IDE (viz podkapitola 5.1), vstupní informace pro algoritmus (podkapitola 5.2) a jeho požadovaný výstup (podkapitola 5.3).

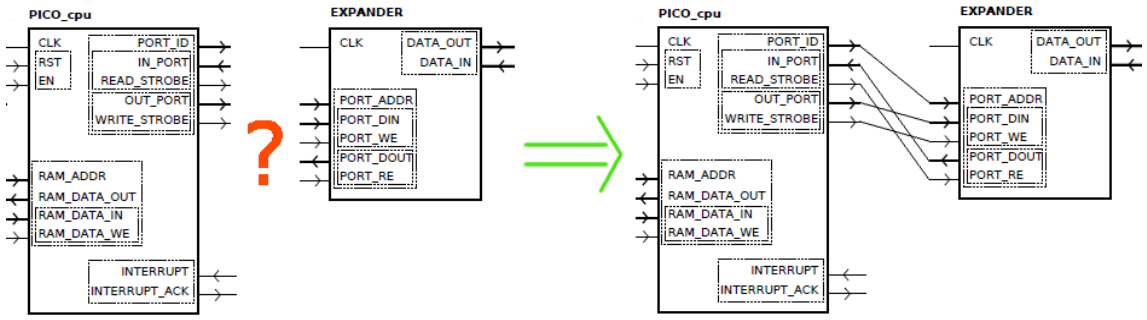
V podkapitole 5.4 bude následovat bližší pohled na jádro algoritmu, jehož nejvýznamnější částí z hlediska návrhu je efektivní hledání nejlepšího párování portů dvou propojovaných komponent, které je teoreticky časově nejnáročnější úlohou při běhu algoritmu. Je proto potřeba hledání nejlepšího propojení komponent formulovat jako problém, který je efektivně řešitelný. V této části kapitola navazuje na kapitolu 4, která se věnovala přiřazovacím problémům a bipartitnímu párování. Popsaná struktura a vlastnosti komponent pak odkazují na kapitolu 2, která se mimo jiné zabývala komponentním modelováním a jazykem Wright, z jehož strukturálního popisu vychází komponentní model užívaný v editoru návrhů ve VLAM IDE.

Na závěr v podkapitole 5.6 shrneme vlastnosti navrženého algoritmu a jeho očekávanou úspěšnost při propojování komponent. Bude následovat návrh grafického uživatelského rozhraní pro spuštění algoritmu a pro dodatečnou specifikaci odvozeného propojení, které bude vloženo do editoru návrhů (v podkapitole 5.7).

## 5.1 Úloha algoritmu

Algoritmus má sloužit uživateli editoru návrhů v prostředí VLAM IDE k usnadnění vytváření schémat zapojení (více o editoru a jeho uživatelském rozhraní bylo uvedeno v sekci 3.3.2). Uživatel v průběhu úpravy schématu využívá paletu nástrojů, kde jsou umístěny komponenty a základní nástroj pro vytváření manuálního propojení jejich portů.

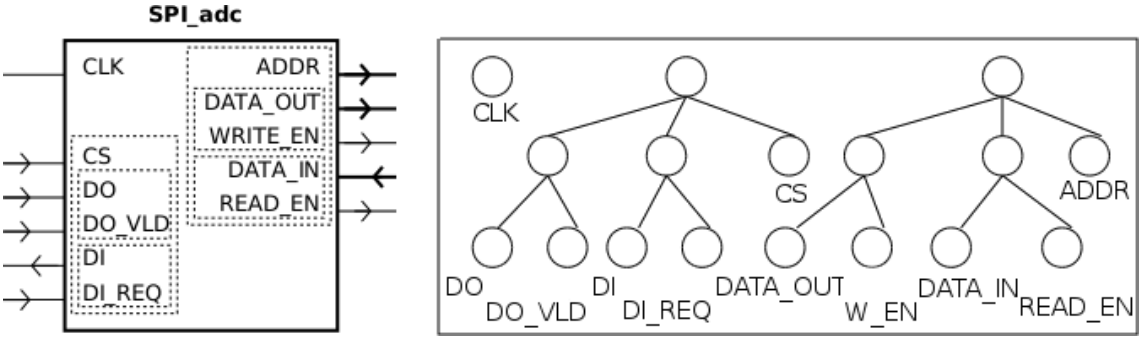
Cílem je vyvinout algoritmus, který uživateli usnadní manuální propojování portů komponent tím, že sám odvodí a navrhne nejlepší propojení. Uživatel pouze vybere dvě komponenty (případně jejich konkrétní rozhraní, což může být i skupina portů), pro které chce toto propojení odvodit. Algoritmus poté navrhne uživateli nejlepší propojení, úlohu algoritmu zjednodušeně ilustruje obrázek 5.1. Algoritmus může obecně odvodit více nejlepších variant řešení, ze kterých uživatel poté vybere tu pro něj nejvhodnější. Variantu navrženou algoritmem poté může uživatel ještě ručně upravit a aplikovat do vytvářeného schématu.




Obrázek 5.1: Inferenční algoritmus odvodí propojení dvou uživatelem zadaných komponent

## 5.2 Vstup algoritmu

Vstupem algoritmu jsou dvě uživatelem vybrané komponenty. V bázi znalostí, která je součástí VLAM IDE, je popsáno rozhraní každé komponenty, to je tvořeno hierarchickou strukturou portů a pinů (listový port šířky jedna), jak naznačuje obrázek 5.2. Každý port má své dané vlastnosti, které mohou určovat jeho kompatibilitu při propojení s jiným portem. Tyto vlastnosti jsme již popsali v sekci 3.3.3.




Obrázek 5.2: Hierarchická struktura portů komponenty SPI_adc (je vynechán kořenový uzel zastupující obsahující komponentu)

Každá vlastnost je zároveň odvozena i pro obsahující nadporty. Pokud port např. obsahuje řídicí a datový port, má on sám také tyto vlastnosti, v případě šířky je šířka nadportu rovna součtu šířek přímých podportů apod. Kompatibilita dvou portů je poté odvozována od směru a typu portů.

## 5.3 Výstup algoritmu

Očekávaným výstupem algoritmu po spuštění na dvou vstupních komponentách je:

- *Nejlepší propojení portů komponent* Několik nejlepších propojení (množin párů listových portů) vstupních komponent seřazených sestupně podle ohodnocení.
- *Rozhodnuté generické parametry pro šířku portů* Určené potřebné šířky pro konkrétní odvozené propojení z dosud nerozhodnutých generických parametrů pro šířky portů.

Uživatel následně z nabídnutých propojení vybere to pro nej nejvhodnější, které nechá aplikovat do vytvářeného schématu. Při takovéto aplikaci nebudou stávající již dříve vytvořená propojení ovlivněna.

## 5.4 Součásti algoritmu

Při prvním pohledu na řešení propojení dvou zadaných komponent zjišťujeme, že při návrhu algoritmu je potřeba řešit několik dílčích problémů.

- *Hledání nejlepšího párování portů* Hledání nejlepšího párování mezi porty dvou vstupních komponent, toto bude stěžejní částí algoritmu určující jeho výslednou použitelnost, neboť může dojít k tzv. *kombinatorické explozi* (více viz sekce 5.4.1).
- *Test kompatibility* Vstupem této elementární operace při propojování jsou vlastnosti dvou portů vstupních komponent, úlohou je určit jejich vzájemnou kompatibilitu (více viz sekce 5.4.2).
- *Ohodnocovací funkce* Pro ohodnocení vzniklého propojení (více viz sekce 5.4.3), na jejím základě dochází k výběru nejlepšího propojení.
- *Výpočet šířek propojení* Bude aplikován v závěru algoritmu pro zjištění využitých kapacit šířek komponent a určení nerozhodnutých generických parametrů pro šířku portů (více viz sekce 5.4.4).

### 5.4.1 Hledání nejlepšího párování portů

Jedná se o teoreticky časově nejnáročnější část algoritmu, kdy hledáme párování mezi dvěmi disjunktními množinami prvků v závislosti na ohodnocení výsledných propojení. Tento problém spadá do množiny kombinatorických problémů a souvisí s problematikou bipartitního párování (viz podkapitola 4.1) a přiřazovacím problémem (viz podkapitoly 4.2 a dále kapitola 4), který je jeho specializací.

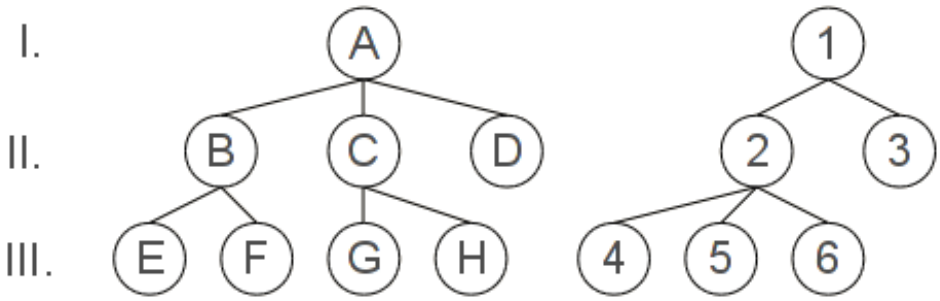
Jak bylo již uvedeno v kapitole 4, algoritmy pro řešení základních podob přiřazování a párování leží ve třídě časové složitosti P, takže je lze rešit exaktním algoritmem pracujícím v polynomiálním čase, zatímco jejich modifikace (kvadratický nebo obecný AP) již zpravidla ve třídě NP. Pro vyvíjený algoritmus by náležitost problému k NP znamenala nutnost zavést algoritmy umělé inteligence pro řešení AP, které byly již dříve zmíněny v podkapitole 4.3 a 4.4.

Při párování dvou stromových struktur portů vstupních komponent (ilustrováno na obr. 5.3) uvažujme dva přístupy, *shora dolů* a *zdola nahoru*. Tyto přístupy se liší nároky, které klademe na používanou ohodnocovací funkci výsledných propojení, čímž rozlišujeme příslušnost řešených problémů ke třídám složitosti P (lineární AP) a NP (kvadratický AP).

#### Shora dolů

Při přístupu *shora dolů* (tedy započetím párování od kořenů stromových struktur) nejprve vytvoříme jednoznačný pár A-1 a poté hledáme rekurzivně páry mezi přímými potomky. Pro pár B-2 vzniklý při dalším sestupu by to znamenalo prohledávání mezi množinami {E, F} a {4, 5, 6}.

Ohodnocovací funkce zde bude určena pouze pro aktuálně uvažovaný pár nezávisle na ostatních párech (a celkové ohodnocení propojení pak bude sumou ohodnocení zúčastněných



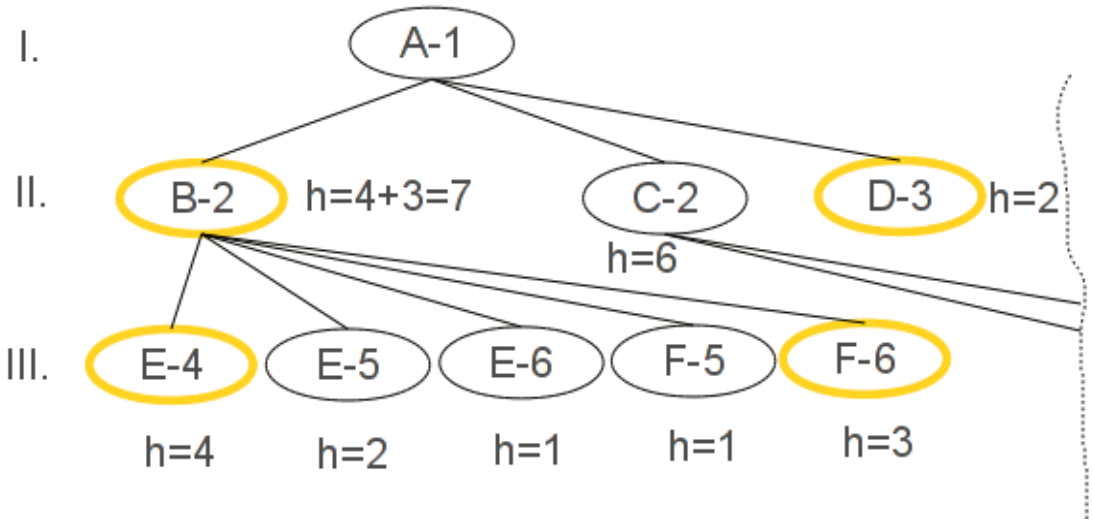

Obrázek 5.3: Příklad struktury dvou vstupních komponent

párů), což nám vytvoří pro každou úroveň stromů problém tzv. maximálního váhovaného bipartitního párování (viz podkapitola 4.8), které je řešitelné v polynomiálním čase.

Základní princip tohoto přístupu rozepišme ve dvou krocích rekurzivního algoritmu:

1. *Synchronní sestup stromovými strukturami* Vytváření možných párů přímých podportů podle výsledku testu jejich kompatibility.
2. *Propagace párů listových portů a jejich hodnocení vzhůru* Výpočet ohodnocení párů a následný výpočet nejlepšího párování podle algoritmu pro maximální váhované bipartitní párování.

Průběh ohodnocování je ilustrován obrázkem 5.4.




Obrázek 5.4: Ilustrace určení hodnoty ohodnocovací funkce $h$ pro vytvářená propojení. Na základě sumy ohodnocení nejlepší kombinace listových párů ohodnotíme i pár přímých předků.

#### Zdola nahoru

Vstupem pro párování by byly přímo listové porty, a zatímco při přístupu shora dolů automaticky nedochází ke křížení portů (nedojde např. ke spárování E-4 a zároveň G-5), při přístupu zdola nahoru by bylo potřeba toto omezit horším ohodnocením celkového propojení obsahujícím takovéto páry.

Zde již ale tímto vstupuje do ohodnocovací funkce závislost ohodnocení jednoho páru na jiných zároveň vytvořených párech (potřebujeme zohlednit míru nežádoucího křížení portů, ke kterému v přístupu shora nedochází), což činí z příbuzného přiřazovacího problému kvadratický (viz podkapitola 4.4), který není řešitelný v polynomiálním čase, a tak by bylo potřeba zavést některou heuristiku nebo jiné metody umělé inteligence.

Výhodou tohoto přístupu by ovšem bylo, že by algoritmus našel i propojení takových listových uzlů, které se nenacházejí ve stejné hloubce (např. H-3).

### 5.4.2 Test kompatibility

Kompatibilita portů dvou komponent je určena vlastnostmi uvedenými v sekci 3.3.3. Nelze například propojit dva výstupní porty, nebo porty různého typu.

Vzhledem k tomu, že vlastnosti jsou odvozeny i pro obsahující nadporty, je možné pro párování shora dolů provádět test kompatibility již při prvním sestupu, a nedochází tak v případě nekompatibility dále k neefektivnímu prohledávání kombinací podportů.

Výsledná kompatibilita dvou portů je potom dána logickým součinem výsledků dílčích testů na kompatibilitu směru, typu a role.

### 5.4.3 Ohodnocovací funkce

Ohodnocovací funkce slouží k ohodnocení výsledného propojení, toto je přirozeně podstatnou informací při výběru nejlepších výsledků. Můžeme ohodnocovat zvlášť jednotlivé kompatibilní páry účastnící se zapojení a výsledné hodnocení určit jako sumu hodnocení těchto párů. Druhou možností je zavést funkci hodnotící obecně n-tici párů a vnést do funkce závislost hodnocení jednoho páru na existenci jiného, to je však již znakem kvadratického přiřazovacího problému (viz podkapitola 4.4).

Veličiny, které můžeme do ohodnocovací funkce zapojit, jsou např.:

1. Propojená šířka portů,
2. obsazenost portů (propojená šířka/šířka portu),
3. rozdíl v hloubce umístění portů ve stromech,
4. vzdálenost rolí (vzhledem k prioritnímu seznamu rolí pro každý port).

Dobré výsledky dávaly již první pokusy, kdy je hodnocení závislé pouze na výsledné šířce propojení. Avšak v některých případech není to nejširší propojení tím hledaným, a tak bude do budoucna potřeba zavést v průběhu testování algoritmu do ohodnocovací funkce i další parametry uvedené výše. Pro výsledný tvar vzorce je možné v budoucnu využít i metod strojového učení.

### 5.4.4 Výpočet šířek propojení

V závěru propojování komponent je potřeba určit využité kapacity jejich šířek a celkovou šířku propojení a určit nerozhodnuté generické parametry pro šířku portů. Toto provedeme ve dvou krocích:

1. *Určení propojené šířky* Urči výslednou obsazenou šířku každého páru propojených portů jako menší z volných kapacit jejich šířek. V případě přítomnosti nerozhodnutého generického parametru pro šířku na některé straně uvažuj z možných hodnot jejich maximum.

2. *Výběr generické šířky* Pro každý pár portů obsahující na některé straně nerozhodnutý generický parametr pro šířku urči minimální potřebanou šířku vzhledem k výsledné šířce z předchozího bodu. V tuto chvíli není řešeno případné zvýšení již rozhodnuté generické šířky po přidání dalšího propojení.

Celková šířka propojení se potom určí jako součet propojených šířek jednotlivých párů portů účastnících se na výsledném propojení.

## 5.5 Navržený algoritmus

Pro výsledný algoritmus zvolíme pro problém efektivního hledání párování portů přístup shora popsaný v sekci 5.4.1. Ten umožní implementovat exaktní algoritmus pracující v polynomiálním čase, který, narozdíl od aproximačních metod strojového učení, vždy nalezne nejlepší řešení. Pro hledání nejlepšího párování bude implementována funkce pro maximální váhované bipartitní párování. Nevýhodou tohoto přístupu je, jak již bylo uvedeno, že při synchronním zanořování ve stromových strukturách komponent nedojde ke spárování portů, které nejsou ve stejné hloubce.

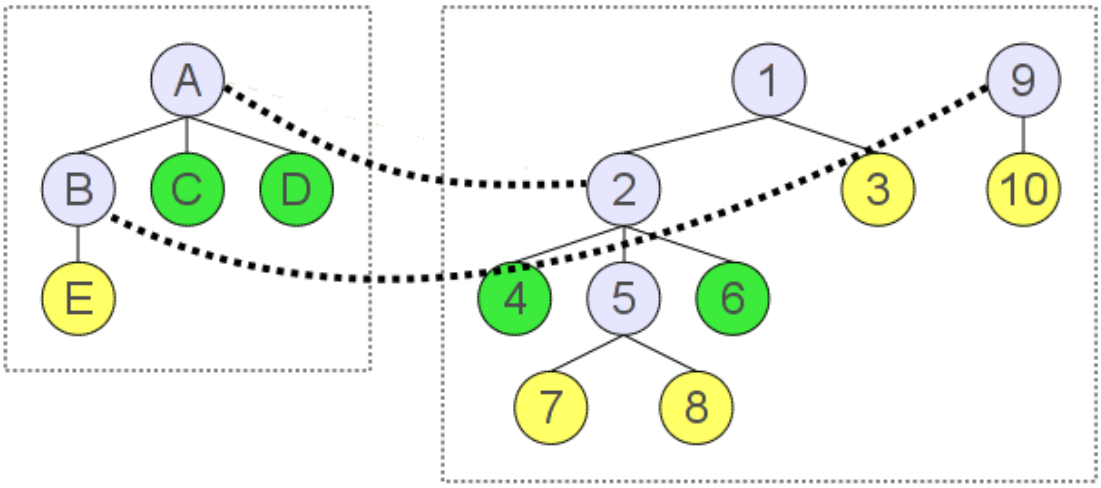
Tuto nevýhodu lze částečně odstranit přidáním inicializační akce algoritmu, *zarovnáním stromů*. Tento krok, který je ilustrován obrázkem 5.5, vytvoří první množinu párů uzlů stromu, z nichž alespoň jeden je přímým poduzlem kořenového uzlu komponenty a oba mají stejnou výšku podstromu. Na takto vytvořených párech se již potom spustí dříve popsaný rekurzivní algoritmus. Výsledným nejlepším propojením je potom nejlépe ohodnocený výsledek těchto dílčích spuštění algoritmu. Tato metoda se nakonec na provedených testech propojení ukázala jako dostačující.

Další části algoritmu, tedy test kompatibility (viz sekce 5.4.2), ohodnocovací funkce (viz sekce 5.4.3) a výpočet šířek propojení (viz sekce 5.4.4) budou implementovány podle návrhů v předchozím textu.

## 5.6 Očekávaná úspěšnost navrženého algoritmu

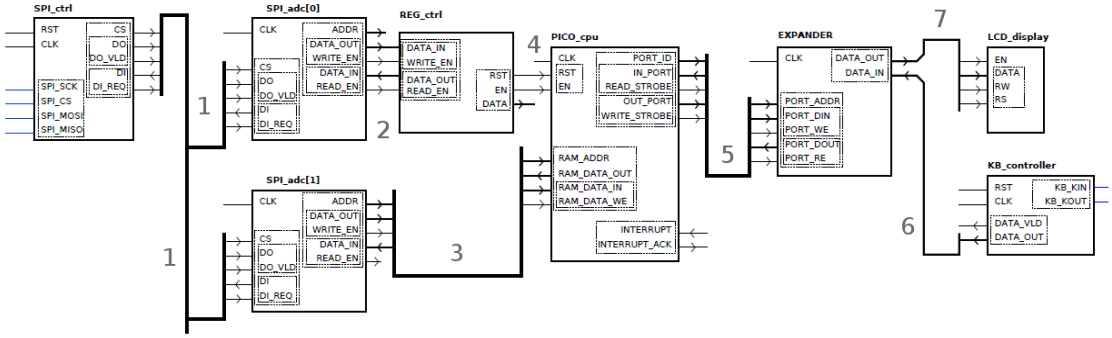
Očekávanou úspěšnost výše navrženého algoritmu, založeného na exaktním řešení maximálního váhovaného bipartitního párování, ilustrujme obrázkem 5.6. Uživatel zde při vytváření schématu zadá algoritmu postupně sedm unikátních dvojic komponent k propojení. Navržený algoritmus je ve své dosavadní formě schopen vyřešit, až na jedno, všechna propojení komponent ve schématu.

Výjimkou je propojení č. 7, kde požadujeme nalezení propojení jednoho portu na více portů druhé komponenty. Pokud bychom chtěli automaticky řešit také tento typ propojení,




Obrázek 5.5: Inicializační krok algoritmu, zarovnání stromů vstupních komponent (spárování portů stromů stejné výšky je znázorněno tečkovanou čarou). Stejnou barvou jsou označeny listové porty vzájemně kompatibilních vlastností. Je vynechán kořenový uzel zastupující obsahující komponentu.

došli bychom ke GAP, který byl popsán v podkapitole 4.3. Tento problém je ovšem opět ve třídě NP, a tak nemůžeme nalézt efektivní exaktní algoritmus.




Obrázek 5.6: PicoBlaze programovatelný z SPI s LCD a klávesnicí

## 5.7 Grafické uživatelské rozhraní pro použití algoritmu

Navržený inferenční algoritmus bude, jak již bylo zmíněno, implementován do grafického editoru návrhů schémat pro FPGA, který je součástí vývojového prostředí VLAM IDE. Popisu tohoto editoru a prostředí byla věnována část kapitoly 3, konkrétně podkapitola 3.3, této technické zprávy.

Zaměřme se nyní na propojení stávající funkčnosti editoru návrhů a jeho grafického uživatelského rozhraní s nově navrženým algoritmem. Je potřeba navrhnout uživatelské rozhraní pro spuštění algoritmu (v sekci 5.7.1) a následnou prezentaci výsledků uživateli,

který vybere výsledek, který mu vyhovuje nejvíce, a pak jej případně ještě ručně upravit (dále v sekcích 5.7.2 a 5.7.3).

### 5.7.1 Spuštění algoritmu

Jak již bylo uvedeno v kapitole 3, editor návrhů se skládá z palety nástrojů a návrhového plátna, na které jsou z palety umisťovány komponenty a poté jsou zde vyznačována jejich vzájemná propojení (nástrojem *Connection*). Jako druhý je v paletě nástrojů předpřipraven nástroj *Algorithm*, určený právě pro spouštění algoritmu, který zde chápeme jako speciální realizaci propojení dvou vybraných komponent.

Po výběru nástroje *Algorithm* a následném označení dvou komponent umístěných v návrhovém plátně, na které má být algoritmus aplikován (případně jejich konkrétních portů/podportů ve stromové hierarchii rozhraní), dojde ke spuštění algoritmu. Výstupem algoritmu jsou návrhy nejlepších způsobů propojení komponent. Tyto návrhy je potřeba uživateli vhodně prezentovat, nejlépe v grafické podobě s naznačením navrhovaných propojení mezi komponentami.

### 5.7.2 Zobrazení výstupu algoritmu uživateli

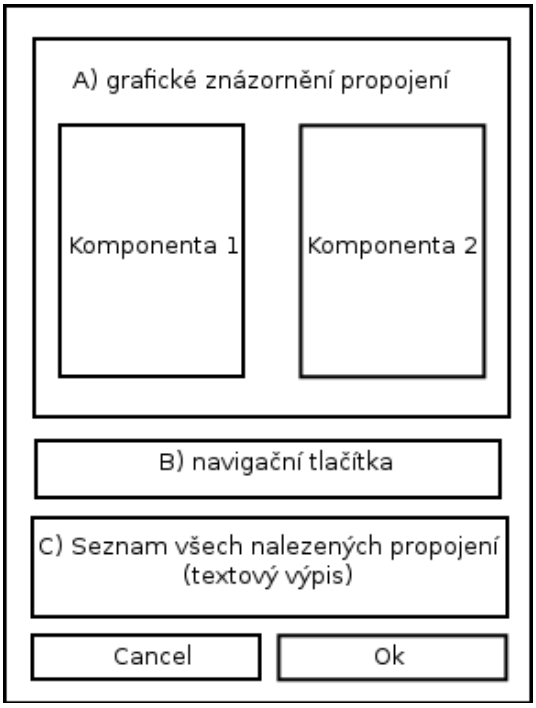
Pro zobrazení mezivýsledků algoritmu navrhneme modální dialog (obr. 5.7), který se uživateli prostředí VLAM IDE zobrazí na popředí a nechá uživatele vybrat pro něj nejvhodnější propojení. Dialog bude obsahovat dvě základní části informující v přehledu a následném detailu o výsledcích algoritmu:

- *Seznam všech nalezených propojení* Dialog ve své dolní části (C) zobrazuje přehledný seznam všech propojení včetně jejich ohodnocení. Textový výpis jednoho propojení je seznam propojení listových portů komponent, kde každá položka seznamu je ve tvaru: `[port1](propŠířka/šířka1)<->(propŠířka/šířka2)[port2]`, kde `port1` a `port2` jsou jména propojených portu první a druhé komponenty, `propŠířka` je šířka těchto portů propojená algoritmem, `šířka1` a `šířka2` celkové šířky propojených portů komponent. V případě, kdy celková šířka byla algoritmem teprve vybrána z přípustných hodnot generického parametru pro šířku, je toto označeno písmenem `G` přidaným k hodnotě celkové šířky. Po vybrání propojení se uživateli zobrazí jeho náhled v grafické části dialogu.

- *Grafické znázornění propojení* Detail jednoho vybraného propojení mezi komponentami (A). V detailu by měly být viditelně označeny porty, které jsou ve schématu již obsazeny, a měly by být graficky odlišeny od portů aktuálně obsazovaných algoritmem. Pro usnadnění orientace by měly být komponenty i jejich porty opatřeny popisky, jak je již zavedeno v editoru schémat.

Dialog bude dále vybaven navigačními tlačítky (B), které umožní přepínat mezi položkami ze seznamu navržených propojení. Aktuálně vybrané propojení může být potvrzeno tlačítkem *Ok*, pak dojde k aplikaci vybraného propojení do schématu, nebo může být celý průběh algoritmu stornován tlačítkem *Cancel*.

Pro usnadnění ovládání zpřístupníme dostupné akce v dialogu i pro ovládání klávesnící, šipky budou ovládat průchod seznamem propojení, klávesy *Enter* a *Escape* budou potvrzovat výběr, resp. stornovat průběh algoritmu.




Obrázek 5.7: Návrh grafického uživatelského rozhraní dialogu pro výběr výsledného propojení

### 5.7.3 Dodatečná úprava navrženého propojení

K dodatečné úpravě propojení ve schématu, které uživatel vybral z předložených návrhů (případně bylo algoritmem vygenerováno jako jediné možné, a následně v dialogu potvrzeno uživatelem), budou sloužit již obvyklé funkce plnohodnotného editoru návrhů. Pro přidání propojení portů do schématu se použije příslušný, již implementovaný, nástroj z palety nástrojů (*Connection*) a pro mazání po výběru propojení portů klávesa *Delete*, případně výběr mazací akce z kontextového menu. Pro upřesnění a editaci jednotlivých propojení mezi porty pro konfiguraci na úrovni jednotlivých pinů, se standardně využije nabídka nastavení z kontextového menu propojení.

# Kapitola 6

# Implementace inferenčního algoritmu a uživatelského rozhraní

Tato kapitola bude věnována popisu implementace inferenčního algoritmu a grafického uživatelského rozhraní pro jeho obsluhu z vývojového prostředí VLAM IDE. Inferenční algoritmus i grafické uživatelské rozhraní byly implementovány podle návrhu v předchozí kapitole 5. V rámci implementace inferenčního algoritmu byla vytvořena i funkce pro maximální váhované bipartitní párování, kterou algoritmus využívá. Algoritmus je dále, jako již dříve v návrhu, rozdělen na specifické součásti, jejichž implementace jsou postupně rozebrány a dále poskládány ve funkční celek, který řeší inferenci propojení komponent. Více o implementaci algoritmu bude popsáno v sekci 6.1.

Implementace grafického uživatelského rozhraní bude popsána v sekci 6.2, z hlediska návrhu se jedná o část, která zajišťuje interakci s uživatelem, který má možnost konečně ovlivnit výsledek inference. Součástí implementace uživatelského rozhraní je také vylepšení funkčnosti stávajícího nástroje *Algorithm*, který umožňuje spuštění algoritmu na vybraných komponentách či konkrétních jejich portech z palety nástrojů v prostředí VLAM IDE.

## 6.1 Implementace inferenčního algoritmu

Inferenční algoritmus je implementován v jazyce Java. Jako nezávislá funkce byl naimplementován i algoritmus pro maximální váhované bipartitní párování (MWBM, více viz sekce 6.1.1), který je při inferenci využíván. Jedná se o originální implementaci tohoto algoritmu v jazyce Java, inspirovanou algoritmickou C++ knihovnou LEDA [18].

Pro aplikaci MWBM na řešení inferenčního problému je potřeba problém formulovat jako bipartitní váhovaný graf. K tomuto slouží pomocné funkce popsané v sekci 6.1.2, jako např. funkce pro definici váhy hrany (ohodnocení propojení).

Výsledný algoritmus pro inferenci propojení komponent je pak popsán v sekci 6.1.3, v této implementaci je již obsaženo např. procházení stromovou strukturou portů dvou propojovaných komponent.

### 6.1.1 Algoritmus pro maximální váhované bipartitní párování

Algoritmus pro MWBM byl realizován na základě zkoumání implementačních detailů tohoto algoritmu v C++ knihovně LEDA. Bylo potřeba naimplementovat potřebné datové struktury (graf, prioritní fronta), případně hledat jejich existující implementace ve volně dostupných knihovnách. Jednodušší datové struktury se daly nalézt přímo ve standardních

třídách balíčku `java.util`. Pro složitější datové struktury (grafy) pak byla použita grafová knihovna JGraphT.

Algoritmus je naimplementován jako na inferenčním algoritmu nezávislá generická funkce, která je umístěna v samostatném balíčku. To umožňuje budoucí využití i v nepříbuzných projektech, je také možné v budoucnu rozšiřovat kód o další grafové funkce. Nyní popíšeme knihovny LEDA a JGraphT a následovat budou implementační detaily algoritmu.

### Algoritmická knihovna LEDA

Knihovna LEDA poskytuje algoritmy pro kombinatorické a geometrické výpočty. Jak již bylo výše zmíněno, je napsána v jazyce C++. Současně s kódem byla vydána i publikace [18], která popisuje teoretická východiska algoritmů a slouží také jako manuál k použití knihovny. Volně dostupná verze knihovny obsahuje pouze základní datové typy. Mimo tuto verzi jsou k dispozici ještě dvě placené verze, akademická a komerční. Doprovodná publikace je ke stažení zdarma na stránkách projektu [1].

### Grafová knihovna JGraphT

JGraphT je knihovna v jazyce Java, která poskytuje implementaci datových struktur a algoritmů teorie grafů. Je distribuována pod licencí LGPL (*GNU Lesser General Public License*). Obsahuje implementace různých typů grafů, jako jsou např. orientované a neorientované grafy, prosté grafy, multigrafy, či grafy s váhovanými hranami. Díky využití generik je knihovna typově bezpečná a je možné její široké použití pro různé instance grafů [2].

### Třídy využité v implementaci algoritmu

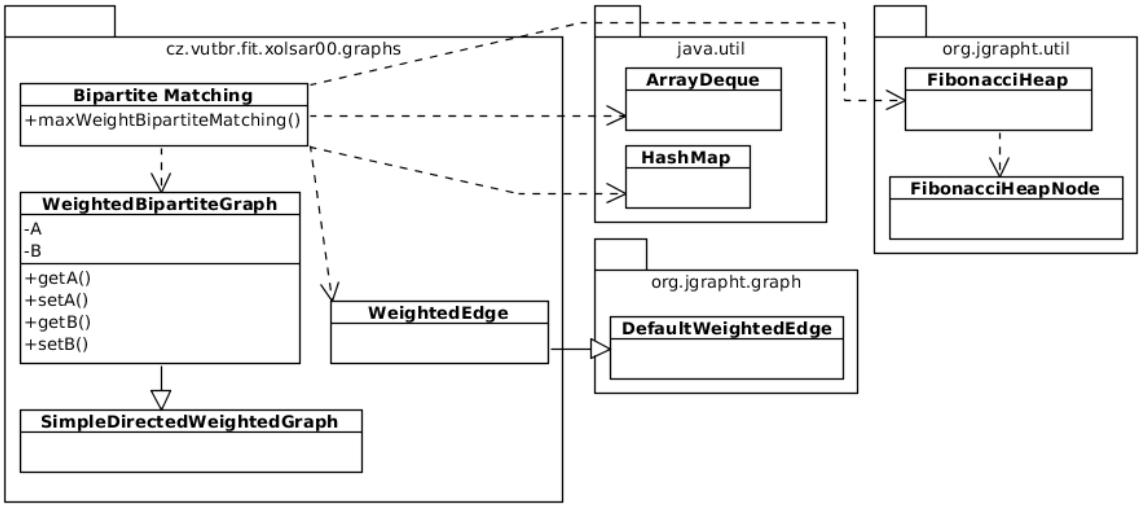
Pro uchovávání informací o uzlech (volnost, potenciál) byla využita hašovací tabulka `HashMap` ze standardního balíčku `java.util`. Jako další z tohoto balíčku byl využit zásobník implementovaný třídou `ArrayDeque`, který je využívaný při hledání nejkratších cest.

Z grafové knihovny JGraphT byly využity třídy pro implementaci grafových struktur. Vytvořený váhovaný bipartitní graf `WeightedBipartiteGraph` byl odvozen od třídy `SimpleDirectedWeightedGraph` doplněním o disjunkci množiny uzlů na dvě podmnožiny $A$ a $B$. Váhovaná hrana je reprezentována třídou `DefaultWeightedEdge`. Pro implementaci prioritní fronty byla využita třída `FibonacciHeap` s prvkem fronty `FibonacciHeapNode`, která je taktéž přístupná v knihovně JGraphT.

### Výsledný algoritmus pro bipartitní párování

Implementovaný algoritmus byl umístěn do balíčku `cz.vutbr.fit.xolsar00.graphs`. Je implementován jako statická metoda třídy `BipartiteMatching` a lze jej zavolat metodou `maxWeightBipartiteMatching` s jediným parametrem, kterým je bipartitní váhovaný graf typu `WeightedBipartiteGraph` popisující instanci párovacího problému. Metoda vrací vybrané párování s maximální váhou reprezentovanou seznamem váhovaných hran `ArrayList<WeightedEdge>`. Diagram tříd implementace algoritmu je znázorněn na obrázku 6.1.

Detaily implementace lze nalézt v programové dokumentaci na přiloženém CD, a tak zde jen stručně zmiňme další implementované třídy používané v algoritmu. Váhovaný bipartitní graf `WeightedBipartiteGraph` byl popsán již výše, jedná se o parametrickou třídu, je tedy možné dosazovat jako uzel grafu libovolný typ v jazyce Java. Třída `WeightedEdge` byla odvozena od `DefaultWeightedEdge` z knihovny JGraphT a slouží pouze pro kratší pojmenování používaných váhovaných hran.




Obrázek 6.1: Diagram tříd implementace maximálního váhovaného bipartitního párování

### 6.1.2 Pomocné funkce inferenčního algoritmu

Implementace inferenčního algoritmu, stejně tak pomocné funkce pro definici inferenčního problému jako instance MWBM, jsou umístěny v balíčku `cz.vutbr.fit.vlam.kb.algorithm.generator`. Diagram tříd implementace celého inferenčního algoritmu včetně pomocných funkcí je znázorněn na obrázku 6.2.

#### Test kompatibility

Test kompatibility slouží k určení, zda v instanci pro MWBM budou porty (resp. uzly v grafu) propojeny, tedy zda mezi nimi bude v grafu hrana, či nikoli. Jak již bylo zmíněno v návrhu algoritmu, test porovnává vlastnosti portů. Implementaci zajišťuje třída `CompatibilityTest`, která má jako parametry konstruktoru dvě rozhraní komponent (typ `Interface`) a po zavolání metody `checkCompatibility` vrací pravdivostní hodnotu.

#### Ohodnocovací funkce

Váhy hran v grafu instance MWBM jsou určeny ohodnocovací funkci umístěnou ve třídě `ConnRanking`, funkce vypočítává ohodnocení na základě vlastností kompatibilních portů podle návrhu a vrací jej jako hodnotu typu `Double`.

#### Výpočet šířek propojení

Metoda `getPluggedWidth` pro výpočet maximální možné šířky propojení dvou portů je umístěna ve třídě `WidthResolver`. Vrací číselnou hodnotu typu `Integer`. Šířka propojení je odvozena od neobsazené šířky obou portů a také bere v úvahu dosud nerozhodnuté generické parametry pro šířku, kdy se snaží využít maximální možnou kapacitu portů. Výsledná šířka je potom užita v ohodnocovací funkci a zároveň se promítá do výsledného návrhu propojení, kdy se automaticky užívá maximální možná šířka propojení.

### 6.1.3 Výsledný inferenční algoritmus

Inferenční algoritmus je implementován metodou `inferConnections` třídy `KbAlgorithm`, jako parametry má 2 množiny portů komponent, jejichž nejlepší propojení hledáme. Výsledek první fáze inference je uchován jako typ `AlgResults`, který je naimplementován jako seznam (`ArrayList` ze standardního balíčku `java.util`) ohodnocených propojení typu `RankedConnection`. `RankedConnection` implementuje rozhraní `Comparable` z balíčku `java.lang`, což umožňuje porovnávání ohodnocených propojení. V druhé fázi, pro výběr již konečného výsledku typu `RankedConnection`, spustí algoritmus grafické uživatelské rozhraní pro uživatelský výběr nejlepšího propojení. Toto rozhraní je implementováno dialogem `FinalResultDialog` v balíčku `cz.vutbr.fit.vlam.kb.generator.dialog`, implementace tohoto dialogu bude blíže popsána v sekci 6.2.2.

Hlavní jádro rekurzivní implementace párování portů vzhledem k jejich stromové struktuře nalezneme ve třídě `ResultNode`, která reprezentuje uzel vytvářeného stromu dvojic portů a dále rekurzivně jejich podportů (v atributu `possibleSubInterConns` typu `ArrayList<ResultNode>`). Zde dochází k volání nezávisle implementovaného algoritmu pro MWBM a jsou zde také použity všechny pomocné funkce, které jsme popsali výše v sekci 6.1.2. Algoritmus používá generický typ `Couple`, který byl implementován pro uchovávání dvojice (tedy dvou prvků, u nichž záleží na pořadí) portů z první a druhé propojované komponenty.

## 6.2 Implementace grafického uživatelského rozhraní

Grafické uživatelské rozhraní pro použití implementovaného inferenčního algoritmu se funkčně skládá ze dvou částí. Nejprve je potřeba algoritmus volat z prostředí VLAM IDE, k čemuž by měl sloužit již nachystaný nástroj Algorithm. Tento však bylo potřeba upravit, aby bylo možné volat algoritmus nejen na dvojici portů, ale také na dvojice port-komponenta a komponenta-komponenta, více o této úpravě bude popsáno v sekci 6.2.1.

Druhou částí implementace grafického uživatelského rozhraní je vytvoření dialogu, který uživateli zobrazí nalezená propojení a nechá jej konečně vybrat to nejlepší, které bude následně aplikováno do schématu. Tam jej pak uživatel již v plně vybaveném editoru může ručně upravit. Více o této části bude popsáno v sekci 6.2.2.

### 6.2.1 Nástroj *Algorithm*

Jak již bylo popsáno v sekci 3.3.2, součástí komponentního editoru návrhu ve VLAM IDE je i paleta nástrojů. Jako jeden z propojovacích nástrojů této palety byl navržen nástroj *Algorithm*, který má sloužit ke spuštění inferenčního algoritmu na dvě uživatelem vybrané komponenty (případně porty). Nástroj je automaticky vytvářen při inicializaci palety v tovární třídě `KbPaletteFactory` (viz obr. 6.3).

Výchozí implementace nástroje *Algorithm* v paletě byla založena na automatickém vytváření propojovacích nástrojů pro modely definované v GMF (odvozené od třídy `ConnectionCreationTool` z balíčku `org.eclipse.gef.tools`). To s sebou přineslo sémantické omezení nástroje, kdy jím nebylo možné vybrat jiné entity, než dva porty, protože propojení (*Connection*) je v modelu definováno právě jen mezi dvěma porty. Po výběru nástroje *Algorithm* z palety byl kurzor myši při přechodu nad komponentami v „zakázaném“ stavu (ikonka označující, že volba není validní), a po dokončení akce nástrojem se tak nebylo možné programově dostat ke dvěma vybraným komponentám (či komponentě a portu),

nad kterými by se algoritmus následně spustil.

Jako nejjednodušší cestou se po zvážení ukázalo být nadefinování nástroje *Algorithm* zcela ve vlastní režii. To obnášelo odvození nástroje od obecné třídy `AbstractTool` a dodefinování funkčnosti po akcích myší (`mouseDown`, `mouseUp`, `mouseDrag`, `mouseMove`), kde bylo nastaveno zapamatování entit, které byly myší v průběhu výběru označeny. Již dříve bylo v původním nástroji *Algorithm* implementováno spuštění algoritmu a aplikace jeho výsledků do schématu, a tak toto nebylo součástí implementace této diplomové práce. Bylo však potřeba rozšířit dohledání konkrétní entity v oblasti vybrané myší, protože obecně se grafický prvek (`EditPart`) přiřazený entitě nachází až hlouběji ve struktuře ostatních grafických prvků, které slouží k zobrazování dalších informací o entitě, jako např. obsazenost portu či popisek, které dříve nástrojem *Algorithm* nebyly zachytitelné.

Součástí implementace bylo i graficky informovat uživatele, zda je jeho aktuální výběr (při přejití myší) validní pro aplikaci algoritmu, či nikoli. K tomu posloužily již definované konstanty pro ikonky z GEF ve třídě `SharedCursors` balíčku `org.eclipse.gef`, `CURSOR_PLUG` a `CURSOR_PLUG_NOT`. Při výběru je také orámováním lehce zvýrazněna aktuální entita, toho bylo dosaženo nastavením fokusu metodou `setFocus()` u grafického prvku entity.

### 6.2.2 Dialog výběru výsledku

Grafický dialog výběru výsledku uživateli zobrazí všechny výsledky, které algoritmus odvodil. Uživatel může mezi výsledky přepínat, nechat si je zobrazovat v grafické části dialogu a nakonec potvrdit výsledek, který je pro něj nejvhodnější. Ten je pak aplikován do schématu. Dialog byl implementován jako součást algoritmu. Ve chvíli, kdy algoritmus dokončí výpočet mezivýsledků a nalezl alespoň jedno možné propojení, zobrazí uživateli dialog (`FinalResultDialog`, viz diagram tříd na obr. 6.4) odvozený od standardního dialogu v SWT[1]. Dialog se zobrazí jako modální dialog na popředí ve VLAM IDE, kdy využívá aktuální vlákno pro uživatelské rozhraní (je použita aktuální instance třídy `Display`). Výslednou podobu implementovaného dialogu lze najít v příloze na obrázcích A.1 a A.2.

Všechny výsledky jsou vypsány ve formě tabulky, výběrem z jejích řádků je možné zobrazovat detail navrženého propojení v grafické části dialogu. Přepínání mezi výsledky je možné pomocí tlačítek umístěných nad tabulkou a také pomocí šipek na klávesnici, stejně tak potvrzení výběru pro aplikaci do schématu (klávesa *Enter*) a opuštění dialogu bez aplikace výsledku (klávesa *Escape*). Grafická část dialogu je vykreslována pomocí knihovny Draw2d (viz podkapitola 1.4). Ta je, jak bylo popsáno, používána z GEF a pro toto použití v dialogu byla dostačující, protože nevyžadujeme možnost uživatelské editace znázorněných prvků. Pro grafickou reprezentaci jednoho vybraného propojení slouží třída `ConnectionFigure`, ta obsahuje grafické znázornění komponent implementované třídou `ComponentFigure`, mezi kterými pak vykresluje vybraná propojení.

#### Textový seznam výsledků

Přehledový seznam všech výsledků je, jak již bylo zmíněno výše, implementován jako tabulka (třída `Table` z balíčku `org.eclipse.swt`), která obsahuje sloupec s ohodnocením nalezených propojení a sloupec s textovým výpisem propojených portů komponent. Textový výpis obsahuje názvy jednotlivých propojených portů, celkovou šířku každého portu a

[1] SWT: The Standard Widget Toolkit, knihovna grafických uživatelských prvků, je používána v prostředí Eclipse [31].

šířku portu, která bude obsazena tímto propojením, podle návrhu ze sekce 5.7.2. Tabulka má definované zachycení události změny vybraného řádku (`SelectionListener` z balíčku `org.eclipse.swt.events`), na tuto změnu se provede překreslení grafického znázornění na nově vybrané propojení.

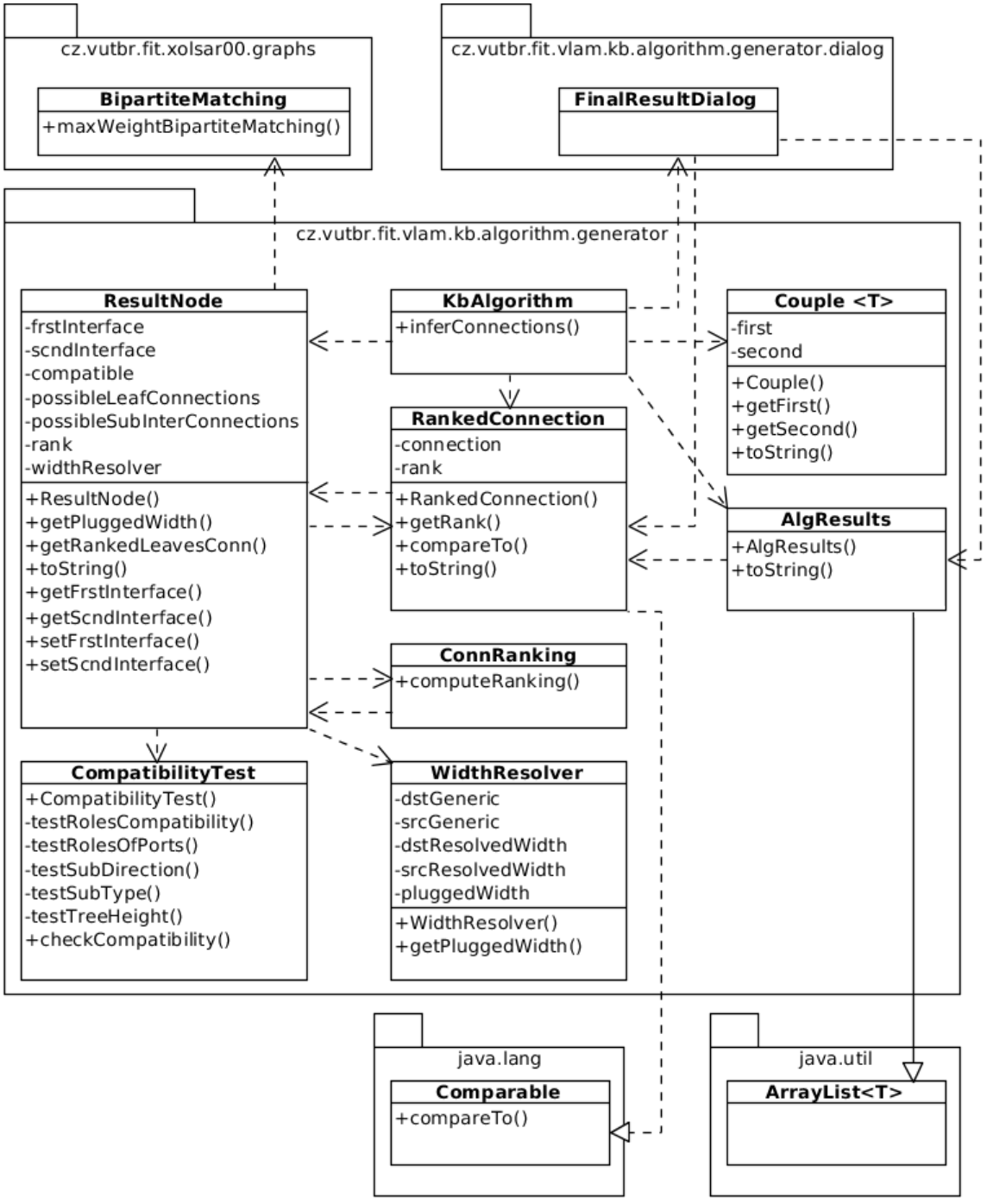
### Grafické znázornění komponent

Každá ze dvou zobrazených komponent (třída `ComponentFigure` odvozena od `RectangleFigure` z knihovny Draw2d) je vykreslena včetně svého rozhraní tvořeného porty, v tomto případě pouze listovými, virtuální porty zde byly abstrahovány. Znázornění struktury komponenty je založeno na definici komponent v bázi znalostí, kde je uchováno, zda se port nachází vlevo, nebo vpravo a v jakém pořadí vzhledem k ostatním portům. Tak bylo možné rekonstruovat vzhled komponent pro snadnější orientaci uživatele. Pro zobrazení směru portů komponenty a jeho obsazenosti v aktuálním schématu byla znovupoužita třída `IORectangle` používaná již v editoru schématu, která definuje vizuální prvek, který je následně umístěn do grafické definice portu pro znázornění těchto vlastností.

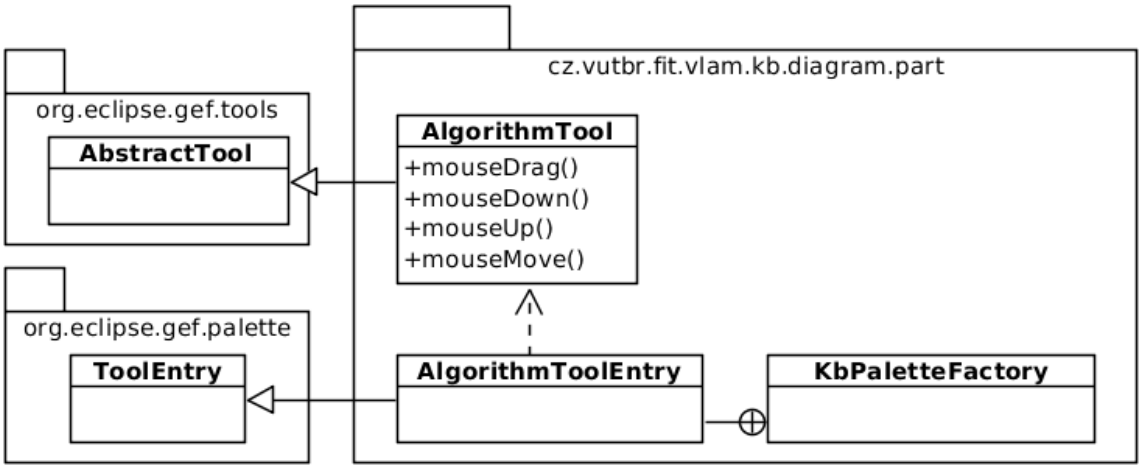
### Grafické znázornění propojení

Po vykreslení obou komponent je procházeno aktuálně vybrané propojení z tabulky a mezi porty komponent jsou vykreslovány propojky, které toto propojení znázorňují. Propojky jsou instancí třídy `PolylineConnection` z knihovny Draw2d. Každá propojka je na obou svých koncích opatřena popiskem vyjadřujícím její šířku. Náročnější částí implementace bylo zajistit, aby nebyly propojky kresleny přes znázorněné komponenty. V knihovně Draw2d jsou pro tyto účely implementovány směrovací (*routing*) algoritmy, nepodařilo se ovšem najít příklad, který by prakticky osvětlil jejich použití, a tak bylo potřeba spíše experimentovat.

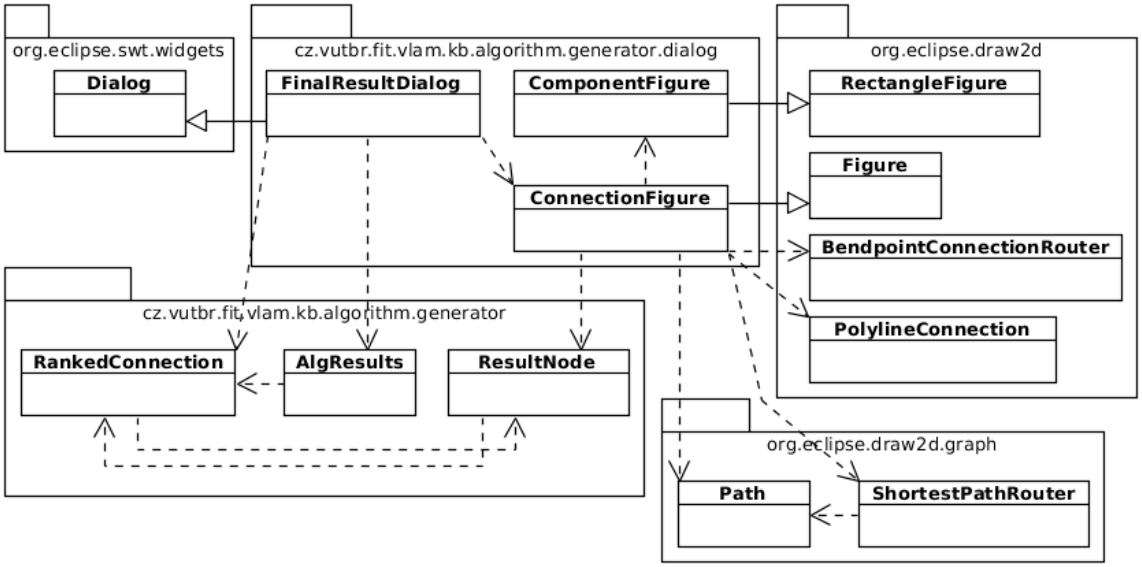
Řešením problému je použití implementace algoritmu nejkratších cest ve třídě `ShortestPathRouter` z balíčku `org.eclipse.draw2d.graph`. Zde je potřeba definovat propojky jako cesty (`Path`), zadat překážky, kterým se tyto cesty mají vyhýbat (v našem případě oblasti komponent) a případně ještě vnější okraj (*margin*), který má být zachován kolem překážek a ostatních cest. Po této definici problému algoritmus vyřeší pro každou cestu body, ve kterých se má cesta ohnout (*bendpoint*), aby se vyhla překážkám. Tyto body ohybu je poté potřeba aplikovat na každou propojku typu `PolylineConnection` a řídit dále její tvar již automaticky pomocí směrovače, který definuje tvar propojky na základě definovaných bodů ohybu, `BendpointConnectionRouter`.




Obrázek 6.2: Diagram tříd inferenčního algoritmu




Obrázek 6.3: Diagram tříd implementace nástroje Algorithm




Obrázek 6.4: Diagram tříd implementace dialogu pro výběr výsledného propojení

# Kapitola 7

# Testování inferenčního algoritmu

Testování vytvořeného inferenčního algoritmu, které bude obsahem této kapitoly, rozdělíme do dvou částí. Implementace algoritmu v sobě zahrnovala programování samostatného algoritmu pro maximální váhované bipartitní párování. Implementace tohoto grafového algoritmu byla popsána v podkapitole 6.1.1, kdy byl implementován jako nezávislá funkce nad parametrickými typy (uzly). Je tedy možné jej testovat jako samostatnou část implementace. Toto testování bude popsáno v podkapitole 7.1.

Dále již bude na zadaných aplikacích z výuky otestován samotný inferenční algoritmus (podkapitola 7.2). Při tomto testování se budeme odkazovat do výsledků popsaných detailněji v příloze B.

## 7.1 Testování algoritmu pro bipartitní párování

Algoritmus pro maximální váhované bipartitní párování, jehož implementace byla popsána v sekci 6.1.1, byl testován s využitím metodologie jednotkového testování (tzv. *unit testing*). Pro tuto metodologii existuje v prostředí Eclipse zásuvný modul JUnit, který automatizuje testování. Více informací o jednotkovém testování a zásuvném modulu JUnit lze najít například v [34].

Definované testy byly umístěny do balíčku `cz.vutbr.fit.xolsar00.graphs.test`. Pro otestování algoritmu byla využita třída `TestCase`, od které jsme odvodili třídu `BipartiteMatchingTest` obsahující metody testující různé případy vstupu. Testovanými vstupy byly různě definované bipartitní grafy s váhovanými hranami, otestovány byly případy, jako je prázdný graf, graf s jednou hranou, či dvěma a více nezávislými, nebo konkurentními hranami (vycházejícími ze stejného uzlu) se stejnou, nebo různou vahou. Ve všech testovaných případech výsledky po spuštění odpovídaly výsledkům očekávaným.

Můžeme tedy konstatovat, že pro testované případy jsme naimplementovali řešení maximálního váhovaného bipartitního párování správně, a tedy s velkou pravděpodobností se jedná i o obecně správné řešení.

## 7.2 Testování inferenčního algoritmu na aplikacích z výuky

Testování algoritmu pro inferenci propojení komponent, jehož implementace jsme popsali v podkapitole 6.1, proběhlo na šesti aplikacích z výuky. Výstupy testování, včetně grafického znázornění a přehledu o úspěšnosti jednotlivých propojení dvojic komponent v rámci jednotlivých aplikací, jsou uvedeny v příloze B. Algoritmus byl spouštěn vždy na celých

komponentách, nikoli pouze na podportech, a výsledná úspěsnost propojení jednotlivých dvojic komponent byla srovnána s hledaným propojením daným zadáním aplikace, a následně roztříděna do pěti skupin podle typu chyby, která případně při odvození nastala. Okomentujeme v následující sekci 7.2.1 souhrnné výsledky, které jsou uvedeny v tabulce v přehledové části B.7 přílohy B. V závěru v sekci 7.2.2 ohodnotíme výsledky vzhledem k podpoře, kterou poskytuje algoritmus uživateli prostředí VLAM IDE.

### 7.2.1 Komentář k výsledkům testování

Z testování, které shrnuje tabulka v příloze oddíl B.7, vyplynulo, že většina hledaných propojení (65,4 %) byla algoritmem nalezena a vrácena z případně více možností jako ta nejlépe ohodnocená. Uživatel tedy může nabídnuté propojení rovnou aplikovat do schématu, aniž by procházel a rozhodoval další možnosti. V ostatních úspěšných případech, které mají zastoupení 11,5 %, bylo hledané propojení sice vráceno, ale již ne jako nejlépe ohodnocené. Uživatel tak musí projít navržená propojení a z nich sám zvolit to nejvhodnější. V tomto případě by výsledky mohlo zlepšit omezení rozsahu výběru vstupu uživatelem označením nikoli celé komponenty, ale pouze některého jejího podportu.

Ve dvou případech (v aplikaci 3 viz příloha oddíl B.3 a 4 viz příloha oddíl B.4), které tvoří 7,7 % z 26 testovaných propojení, nebyla inference propojení úspěšná z důvodu nekompatibility zadaných typů portů. Konkrétně se jednalo o typy *control* a *data*. V tomto případě by pomohlo zmírnit hodnocení kompatibility těchto dvou typů v testu, případně zavést smíšené typy portů.

V jednom případě (v aplikaci 1 viz příloha oddíl B.1) jsme při testování narazili na situaci, kdy není možné najít celé hledané propojení, ale pouze jeho část z důvodu omezení definicí bipartitního párování. Není totiž možné propojit jeden port na více portů zároveň. Toto jsme diskutovali již v podkapitole 5.6 o očekávané úspěšnosti algoritmu.

Ve třech případech (tj. 11,5 %) bylo hledané propojení pouze podmnožinou nalezeného propojení. Tento jev je důsledkem definice maximálního váhovaného párování, kdy se snažíme vždy nalézt maximální řešení. Uživatel může toto chování opět omezit aplikací algoritmu pouze na podporty místo celých komponent. Samozřejmě, vždy má uživatel následně možnost nechtěná propojení v editoru schémat ručně vymazat, či naopak přidat další žádaná propojení.

### 7.2.2 Ohodnocení úspěšnosti algoritmu

Ohodnocení celkové úspěšnosti algoritmu můžeme učinit z hlediska nápovědy, kterou algoritmus poskytuje uživateli VLAM IDE. Rozdělme tedy výsledek do tří skupin:

1. Uživatel je algoritmem naveden na nejlepší řešení (65,4 %).
2. Vyžaduje se sofistikovaná spolupráce uživatele při výběru konečného výsledku, případně již při úvodním výběru konkrétních podportů pro spuštění algoritmu (23 %).
3. Výsledek není ani při nápovědě od uživatele nalezen (11,5 %).

Některé výsledky z bodu 2 je možné vylepšit úpravou definice ohodnocovací funkce, jejímiž možnými parametry (mimo aktuálně použité šířky) jsme se zabývali v sekci 5.4.3. Můžeme se tak zaměřit na zvýraznění některých vlastností hledaných propojení, které by se následně promítly do lepšího výsledku ohodnocovací funkce. Tímto postupem však zřejmě nelze odstranit 11,5 % případů obsažených v tomto bodě, kdy je hledané propojení pouze

podmnožinou nalezeného. Pro tyto speciální případy by mohla být báze znalostí (KB) rozšířena o expertní pravidla zakázaných a povolených propojení.

# Závěr

Diplomová práce se zabývala návrhem algoritmu pro inferenci propojení hardwarových komponent. Algoritmus je určen pro použití v editoru návrhu schémat pro FPGA čipy, který je součástí školního vývojového prostředí VLAM IDE. V rámci práce byla nejdříve nastudována vývojová platforma Eclipse, na které je prostředí VLAM IDE založeno, a jejíž porozumění tak bylo základem pro úspěšné vytvoření algoritmu podporujícího návrh schémat v tomto prostředí. Dalšími položkami v seznamu potřebného teoretického základu byly metody návrhu vestavěných systémů, zejména pak ty, které jsou uplatňovány ve výuce, a seznámení se s projektem Virtuální laboratoře mikroprocesorové techniky (VLAM), v rámci kterého byl inferenční algoritmus řešen.

Pro úspěšný návrh inferenčního algoritmu bylo potřeba detailně nastudovat příbuznou problematiku přiřazovacích a párovacích problémů. Z tohoto průzkumu vyplynuly možnosti a omezení realizace algoritmu, které byly podrobně diskutovány, a na základě kterých byla navržena výchozí podoba řešení. Navazující implementace navrženého algoritmu byla následována návrhem a realizací grafického uživatelského rozhraní pro obsluhu algoritmu z prostředí VLAM IDE. Kombinací stávajících prostředků VLAM IDE a nového rozhraní pro inferenční algoritmus vznikl intuitivně ovladatelný nástroj, který uživateli napovídá správná propojení zadaných komponent, a tím mu usnadňuje práci při navrhování hardwarových schémat.

Implementovaný algoritmus byl dále testován na zadaných aplikacích z výuky. Testováním se odhalily, či jen potvrdily, nevýhody návrhu, kdy algoritmus v této své aktuální podobě nebyl schopen některá propojení odvodit správně. Některá tato kritická místa implementace lze však efektivně řešit vylepšeními, která byla navržena v návaznosti na provedené testy. Další vývoj algoritmu se tak může ubírat úpravou ohodnocovací funkce, či přidáním dalších typů informací do používané báze znalostí.

První verze návrhu algoritmu byla publikována na konferenci DATAKON 2010 [9], návrh řešení vycházející ze širších teoretických poznatků pak na konferenci Student EEICT 2011 [21]. Inferenčnímu algoritmu byla věnována i podkapitola knihy o projektu VLAM [6], která byla vydána v roce 2011.

# Literatura


[1] Algorithmic Solutions Software GmbH: LEDA 6.3. [online], cit. 2012-06-08.
URL http://www.algorithmic-solutions.com/leda/index.htm

[2] Barak Naveh a přispěvatelé: JGraphT. [online], cit. 2012-06-08.
URL http://jgrapht.org/

[3] Bertels, P.; D'Haene, M.; Degryse, T.; aj.: Teaching skills and concepts for embedded systems design. *SIGBED Rev.*, ročník 6, January 2009: s. 4:1–4:8, ISSN 1551-3688, doi:http://doi.acm.org/10.1145/1534480.1534484.
URL http://doi.acm.org/10.1145/1534480.1534484

[4] Burkard, R.; Dell'Amico, M.; Martello, S.: *Assignment problems*. SIAM, Society for Industrial and Applied Mathematics, 2009, ISBN 9780898716634.

[5] Clayberg, E.; Rubel, D.: *Eclipse Plug-ins*. Addison-Wesley Professional, třetí vydání, 2008, ISBN 0321553462, 9780321553461.

[6] Dulík, T.; Křivka, Z.; Kadlec, J.; aj.: *Virtuální laboratoř pro vývoj aplikací s mikroprocesory a FPGA*. Akademické nakladatelství CERM, 2011, ISBN 978-80-7204-754-3.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9727

[7] Fakulta informačních technologií VUT v Brně: Hardware/Software Codesign (stránky předmětu). [online], cit. 2012-06-14.
URL http://www.fit.vutbr.cz/study/courses/index.php?id=7974

[8] International Business Machines Corp.: Introducing the GMF Runtime. [online], cit. 2011-12-16.
URL http://eclipse.org/articles/Article-Introducing-GMF/article.html

[9] Jirák, O.; Křivka, Z.; Olšarová, N.; aj.: Odvozování propojení komponent pro podporu návrhu pro malé FPGA čipy. In *DATAKON 2010 Proceedings (Ed. Petr Šaloun)*, University of Ostrava, 2010, ISBN 978-80-7368-424-2, s. 81–90.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9374

[10] Jirák, O.; Křivka, Z.; Vašíček, Z.: Integrated Development Environment for Virtual Laboratory. In *International Technology, Education and Development Conference*, International Association for Technology, Education and Development, 2011, ISBN 978-84-614-7423-3, str. 10.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9503

[11] Khan, L.; Jeong, T. T.; Park, G.; aj.: A HW/SW Co-design Methodology: An Accurate Power Efficiency Model and Design Metrics for Embedded System. In *Proceedings of the 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, SNPD '09, Washington, DC, USA: IEEE Computer Society, 2009, ISBN 978-0-7695-3642-2, s. 3–7, doi:http://dx.doi.org/10.1109/SNPD.2009.71.
URL http://dx.doi.org/10.1109/SNPD.2009.71

[12] Křivka, Z.; Masopust, T.: *Grafové algoritmy (materiály k přednáškám)*. FIT VUT v Brně, 2010.

[13] Loiola, E. M.; de Abreu, N. M. M.; Boaventura-Netto, P. O.; aj.: A survey for the quadratic assignment problem. *European Journal of Operational Research*, ročník 176, 2007: s. 657–690.

[14] Lorena, L. A. N.; Narciso, M. G.: Relaxation heuristics for a generalized assignment problem. *European Journal of Operational Research*, ročník 91, č. 3, 1996: s. 600–610.
URL
http://econpapers.repec.org/RePEc:eee:ejores:v:91:y:1996:i:3:p:600-610

[15] Lu, S.; Halang, W.: Platform-Independent Specification of Component Architectures for Embedded Real-Time Systems Based on an Extended UML. In *Component-Based Software Development for Embedded Systems*, ročník 3778, editace C. Atkinson; C. Bunse; H.-G. Gross; C. Peper, Springer Berlin / Heidelberg, 2005, ISBN 978-3-540-30644-3, s. 123–142, 10.1007/11591962.7.
URL http://dx.doi.org/10.1007/11591962.7

[16] Malinowski, A.; Yu, H.: Comparison of Embedded System Design for Industrial Applications. *IEEE Trans. Industrial Informatics*, ročník 7, č. 2, 2011: s. 244–254.

[17] Masařík, K.: Jazyky pro popis architektury. Technická zpráva, Ústav informačních systémů, FIT VUT.
URL http:
//www.fit.vutbr.cz/research/pubs/TR/2006/sem_uifs/s060306slidy1.pdf

[18] Mehlhorn, K.; Näher, S.: *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999, ISBN 0-521-56329-1.

[19] Meyer-Baese, U.; Botella, G.; Castillo, E.; aj.: A Balanced HW/SW Teaching Approach for Embedded Microprocessors. *International Journal of Engineering Education*, ročník 26, č. 3, 2010: s. 584–592.

[20] Moore, W.; Dean, D.; Gerber, A.; aj.: *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. IBM Corp., 2004, ISBN 0738453161.

[21] Olšarová, N.: Component Interconnection Inference Algorithm. In *Proceedings of the 17th Conference STUDENT EEICT 2011*, Brno University of Technology, 2011, ISBN 978-80-214-4272-6.

[22] Olšarová, N.: Inference propojení komponent, semestrální projekt. Technická zpráva, Brno, FIT VUT v Brně, 2012.

[23] OSGi$^{TM}$ Alliance: OSGi Technology. [online], cit. 2011-12-16.
URL http://www.osgi.org/About/Technology

[24] Pentico, D. W.: Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, ročník 176, č. 2, 2007: s. 774–793.

[25] Robert Allen, D. G.: The WRIGHT architectural specification language. Technická zpráva, Carnegie Mellon University, School of Computer Science, Pittsburgh, 1996.

[26] Rychlý, M.: Formální specifikace architektur informačních systémů. Technická zpráva, Ústav informačních systémů, FIT VUT.
URL
http://www.fit.vutbr.cz/study/courses/VPD/public/0506VPD-Rychly.pdf

[27] Rychlý, M.: *Formal-based Component Model with Support of Mobile Architecture*. Dizertační práce, 2010.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9174

[28] Sahni, S.; Gonzalez, T. F.: P-Complete Approximation Problems. *J. ACM*, ročník 23, č. 3, 1976: s. 555–565.

[29] Sun, F.; Li, X.; Wang, Q.; aj.: FPGA-based embedded system design. In *Circuits and Systems, APCCAS 2008. IEEE Asia Pacific Conference*, 2008, ISBN 978-1-4244-2342-2, s. 733 – 736, doi:10.1109/APCCAS.2008.4746128.

[30] The Eclipse Foundation: Programmer's Guide. [online], cit. 2011-12-15.
URL http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/int_eclipse.htm

[31] The Eclipse Foundation: SWT: The Standard Widget Toolkit. [online], cit. 2012-07-25.
URL http://www.eclipse.org/swt/

[32] UNIS, UTIA, FIT: VLAM, Virtuální laboratoř aplikace mikroprocesorové techniky. [online], cit. 2012-06-06.
URL http://www.vlam.cz/

[33] Vašíček, Z.: FITkit. [online], cit. 2012-06-06.
URL http://merlin.fit.vutbr.cz/FITkit/

[34] Vogel, L.: JUnit - Tutorial . [online], cit. 2012-07-30.
URL http://www.vogella.com/articles/JUnit/article.html

[35] Özbakir, L.; Baykasoğlu, A.; Tapkan, P.: Bees algorithm for generalized assignment problem. *Applied Mathematics and Computation*, ročník 215, č. 11, 2010: s. 3782 – 3795, ISSN 0096-3003, doi:10.1016/j.amc.2009.11.018.
URL http://www.sciencedirect.com/science/article/pii/S0096300309010078

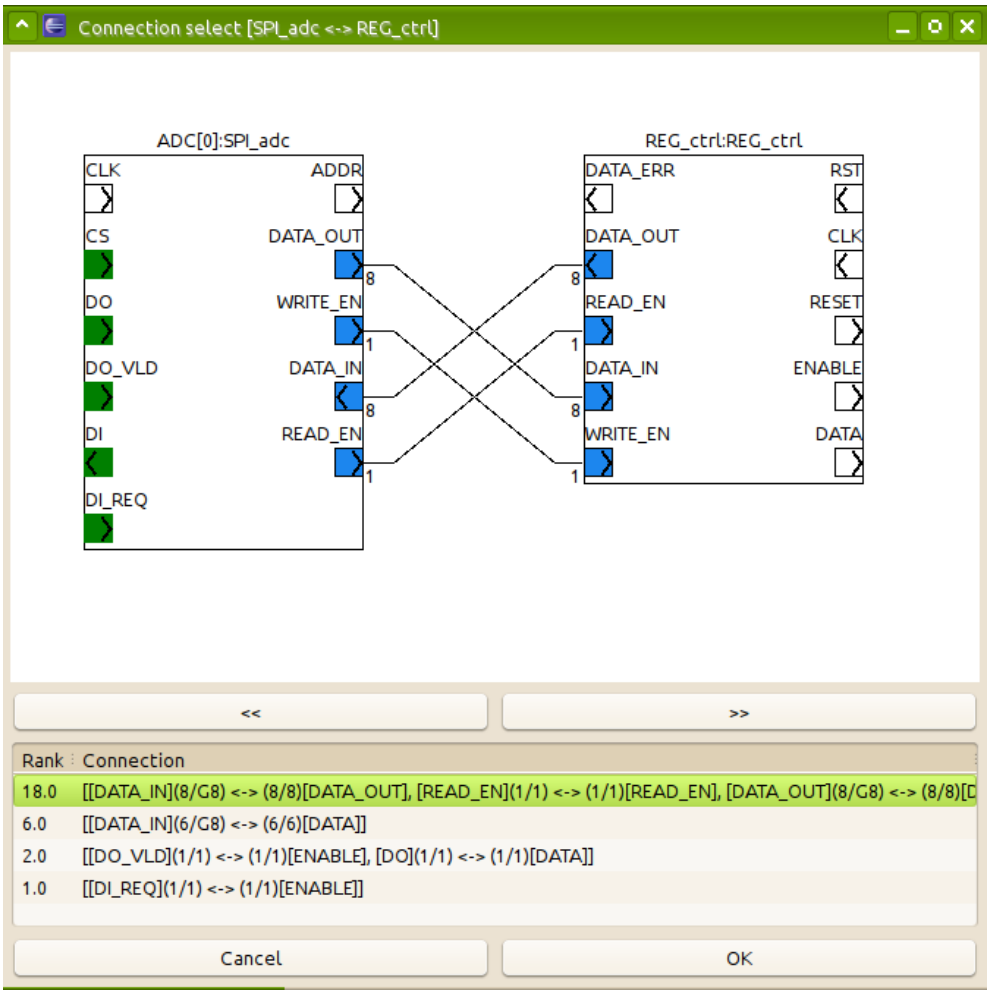
# Seznam použitých zkratek

| | |
|---|---|
| ADL | Architecture Description Language |
| AP | Assignment Problem |
| API | Application Programming Interface |
| ASIC | Application-Specific Integrated Circuit |
| ASIP | Application-Specific Instruction-Set Processor |
| ASL | Architectural Specification Language |
| BM | Bipartite Matching |
| BPM | Business Process Model |
| CBD | Component Based Development |
| CBS | Component Based System |
| CPU | Central Processing Unit |
| CSP | Communicating Sequential Processes |
| DFD | Data Flow Diagram |
| DSP | Digital Signal Processor |
| EMF | Eclipse Modeling Framework |
| EPL | Eclipse Public License |
| FPGA | Field Programmable Gate Array |
| GAP | Generalized Assignment Problem |
| GEF | Graphical Editing Framework |
| GMF | Graphical Modeling Framework |
| GUI | Graphical User Interface |
| HDL | Hardware Description Language |
| IDE | Integrated Development Environment |

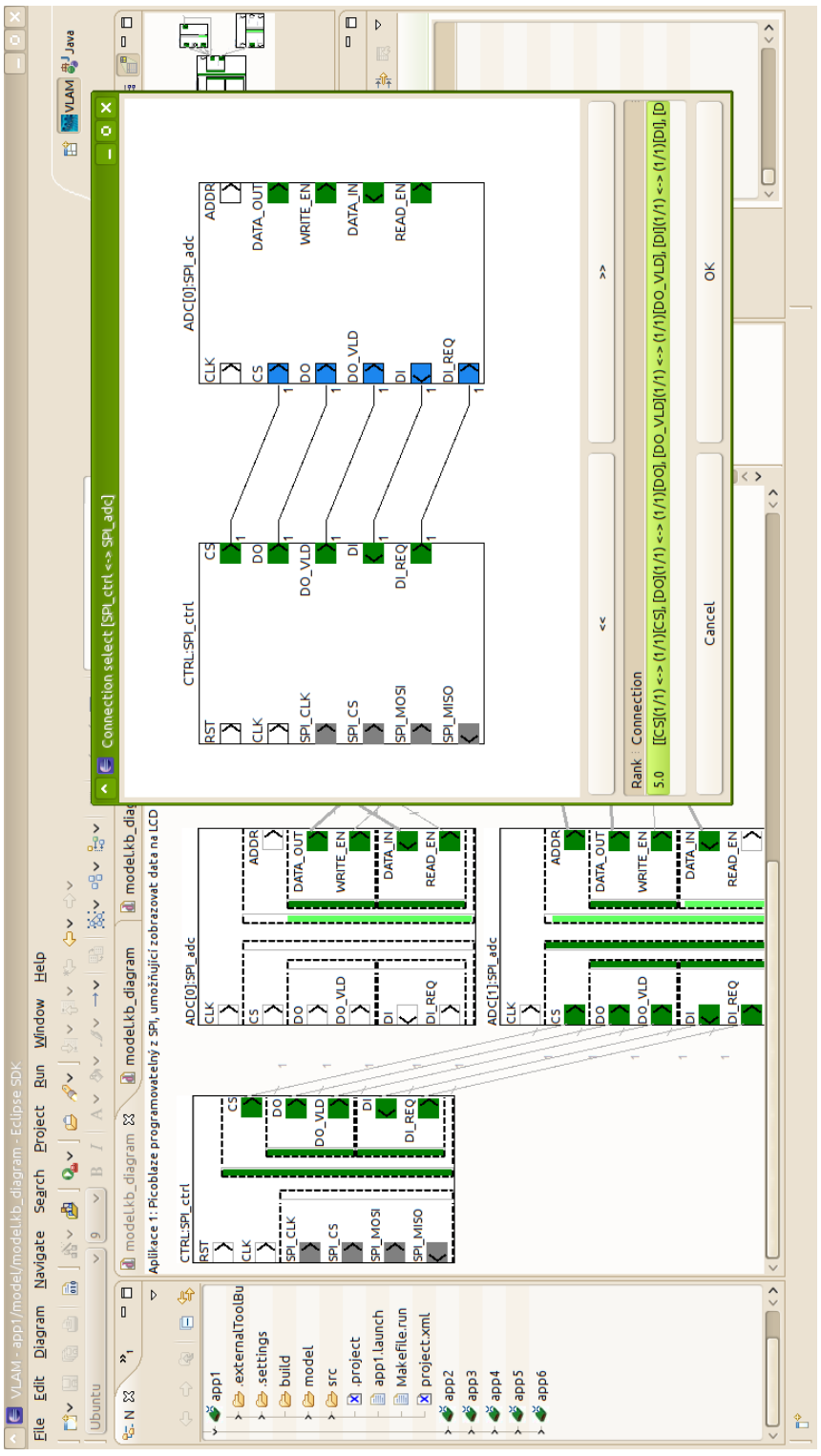
| | |
|---|---|
| KB | Knowledge Base |
| KPN | Kahn Process Network |
| LAP | Linear Assignment Problem |
| MCU | Multi-Point Control Unit |
| MDA | Model Driven Architecture |
| MVC | Model-View-Controller |
| MWBM | Maximum Weight Bipartite Matching |
| MWM | Maximum Weight Matching |
| OMG | Object Management Group |
| OS | Operating System |
| QAP | Quadratic Assignment Problem |
| RAM | Random Access Memory |
| ROM | Read-Only Memory |
| UI | User Interface |
| UML | Unified Modeling Language |
| USB | Universal Serial Bus |
| VHDL | VHSIC Hardware Description Language |
| VHSIC | Very-High-Speed Integrated Circuits |
| VLAM | Virtual Laboratory of Microprocessor Technology Application |
| XML | eXtensible Markup Language |

# Příloha A

# Implementace grafického uživatelského rozhraní




Obrázek A.1: Implementované uživatelské rozhraní (dialog) pro výběr výsledného propojení




Obrázek A.2: VLAM IDE s dialogem pro výběr výsledného propojení na popředí

# Příloha B

# Testování na aplikacích z výuky

Výsledky testování na zadaných aplikacích ve formě tabulky. Každá aplikace byla otestována postupným propojováním dvojic komponent inferenčním algoritmem (*komponenta 1*, *komponenta 2*), *pořadí* označuje, na kterém místě se hledané propojení umístilo vzhledem k ohodnocovací funkci z nalezených možných propojení *celkem*, *úspěch* algoritmu je značen symbolem:

- A Zcela úspěšné odvození, bylo nalezeno hledané propojení, které bylo algoritem vráceno jako nejlépe ohodnocené.
- B Úspěšné odvození, hledané propojení bylo vráceno jako jedna z možností, která ale nebyla nejlépe ohodnocena.
- C Hledané propojení nebylo nalezeno kvůli konfliktu typu portů, které jsou součástí hledaného propojení.
- P Propojení bylo nalezeno jen částečně kvůli omezení párovacího algoritmu (nelze propojit jeden port na více portů najednou).
- S Hledané propojení je podmnožinou nalezeného.

## B.1 Aplikace 1

Picoblaze programovatelný z SPI, umožňující zobrazovat data na LCD a reagovat na stisk kláves na klávesnici, detail propojení viz obr. B.1.

| komponenta 1 | komponenta 2 | pořadí | celkem | úspěch |
|---|---|---|---|---|
| CTRL | ADC[0] | 1 | 1 | A |
| CTRL | ADC[1] | 1 | 1 | A |
| ADC[0] | REG_ctrl | 1 | 4 | A |
| ADC[1] | CPU | 1 | 6 | A |
| REG_ctrl | CPU | 3 | 3 | B |
| CPU | Expander | 2 | 6 | B |
| Expander | LCD | 1 | 3 | P |
| Expander | KB_ctrl | 2 | 3 | B |

## B.2 Aplikace 2

Obsluha klávesnice z mikrokontroleru, detail propojení viz obr. B.2.

| komponenta 1 | komponenta 2 | pořadí | celkem | úspěch |
|---|---|---|---|---|
| CTRL | ADC | 1 | 1 | A |
| ADC | KB_ctrl | 1 | 3 | A |

## B.3 Aplikace 3

Obsluha klávesnice z mikrokontroleru pomocí přerušení, detail propojení viz obr. B.3.

| komponenta 1 | komponenta 2 | pořadí | celkem | úspěch |
|---|---|---|---|---|
| CTRL | ADC[kb] | 1 | 1 | A |
| CTRL | ADC[irq] | 1 | 1 | A |
| ADC[kb] | KB_ctrl | 1 | 3 | A |
| ADC[irq] | IRQ | 1 | 3 | A |
| KB_ctrl | IRQ | - | 2 | C |

Propojení KB_ctrl a IRQ konflikt typů portů (*data*/*control*), nalezena dvě jiná propojení.

## B.4 Aplikace 4

Obsluha PS/2 klávesnice z mikrokontroleru, detail propojení viz obr. B.4.

| komponenta 1 | komponenta 2 | pořadí | celkem | úspěch |
|---|---|---|---|---|
| CTRL | ADC[kb] | 1 | 1 | A |
| CTRL | ADC[irq] | 1 | 1 | A |
| ADC[kb] | PS2 | 1 | 1 | A |
| ADC[irq] | IRQ | 1 | 3 | A |
| IRQ | PS2 | - | 1 | C |

Propojení IRQ a PS2 konflikt typů portů (*data*/*control*), nalezeno jedno jiné propojení.

## B.5 Aplikace 5

Obsluha PS/2 klávesnice z PicoBlaze, ovládání LED, detail propojení viz obr. B.5.

| komponenta 1 | komponenta 2 | pořadí | celkem | úspěch |
|---|---|---|---|---|
| CPU | REG | 2 | 2 | S |
| CPU | REG_1 | 2 | 2 | S |
| REG | PS2 | 1 | 1 | A |
| REG_1 | LED | 1 | 1 | A |

## B.6 Aplikace 6

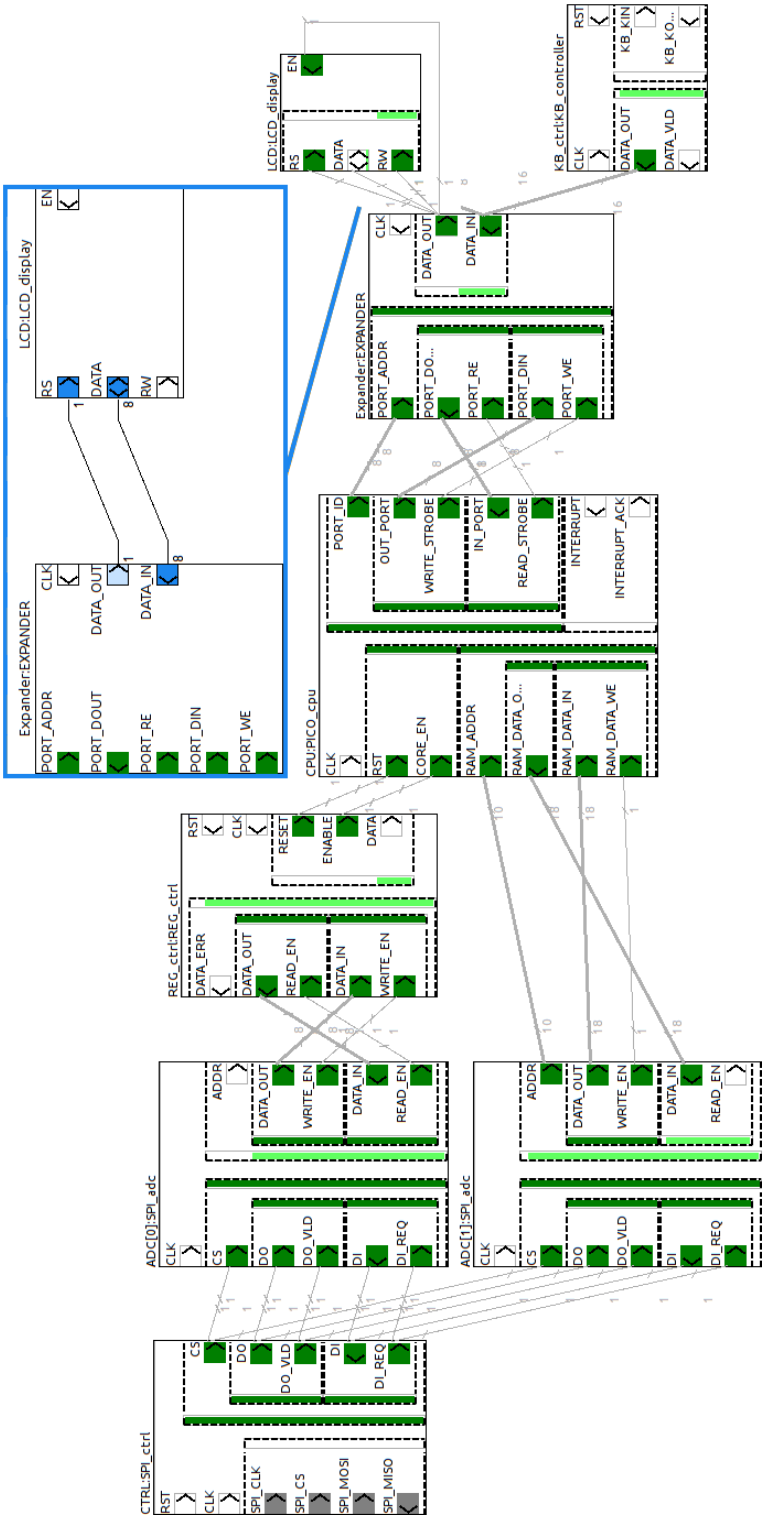
Připojení interní paměti BRAM k SPI, detail propojení viz obr. B.6.

| komponenta 1 | komponenta 2 | pořadí | celkem | úspěch |
|---|---|---|---|---|
| CTRL | ADC | 1 | 1 | A |
| ADC | RAM | 1 | 3 | S |

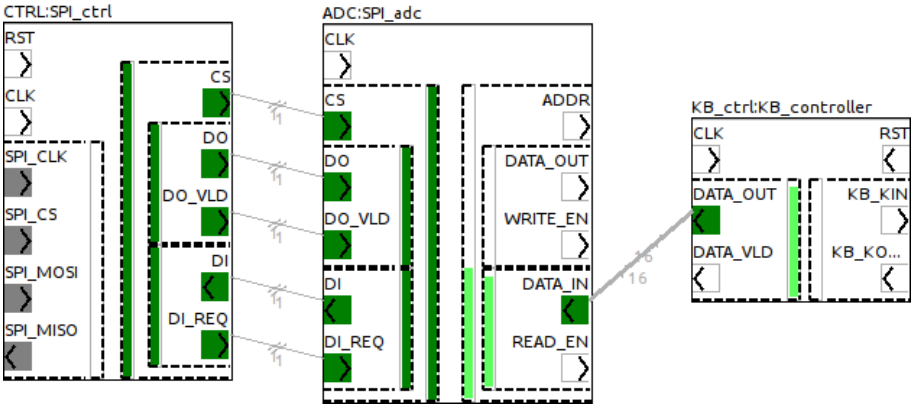
## B.7 Souhrn výsledků

Souhrnná tabulka *aplikací*, na kterých byl algoritmus testován, je udán celkový počet *propojení* v aplikaci, úspěch při propojování a procentuální zastoupení úspěšnosti zaokrouhleno na jedno desetinné místo.

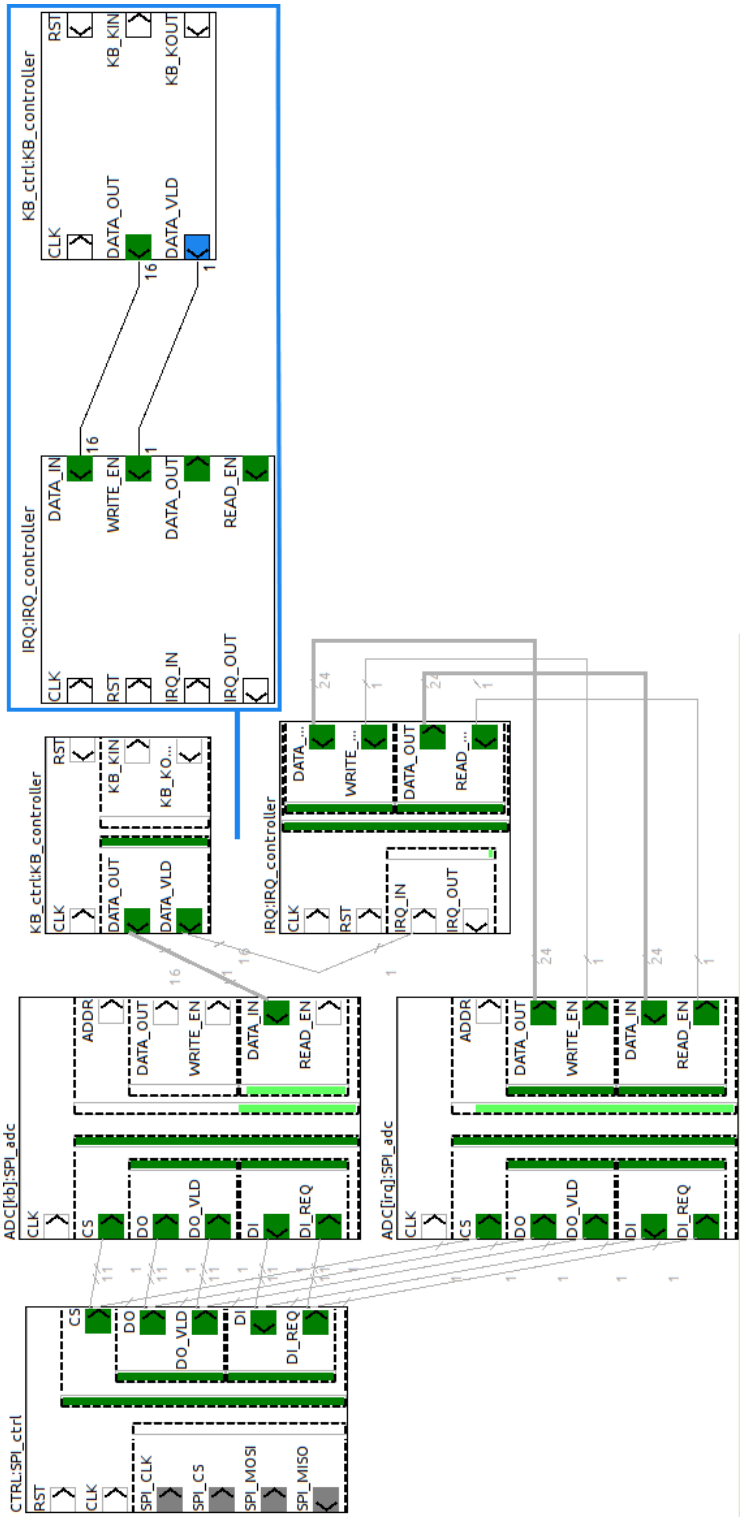
| **Aplikace** | **Propojení** | **A** | **B** | **C** | **P** | **S** |
|---|---|---|---|---|---|---|
| 1 | 8 | 4 | 3 | - | 1 | - |
| 2 | 2 | 2 | - | - | - | - |
| 3 | 5 | 4 | - | 1 | - | - |
| 4 | 5 | 4 | - | 1 | - | - |
| 5 | 4 | 2 | - | - | - | 2 |
| 6 | 2 | 1 | - | - | - | 1 |
| Celkem | 26 | 17 | 3 | 2 | 1 | 3 |
| Procent | | 65,4 % | 11,5 % | 7,7 % | 3,8 % | 11,5 % |



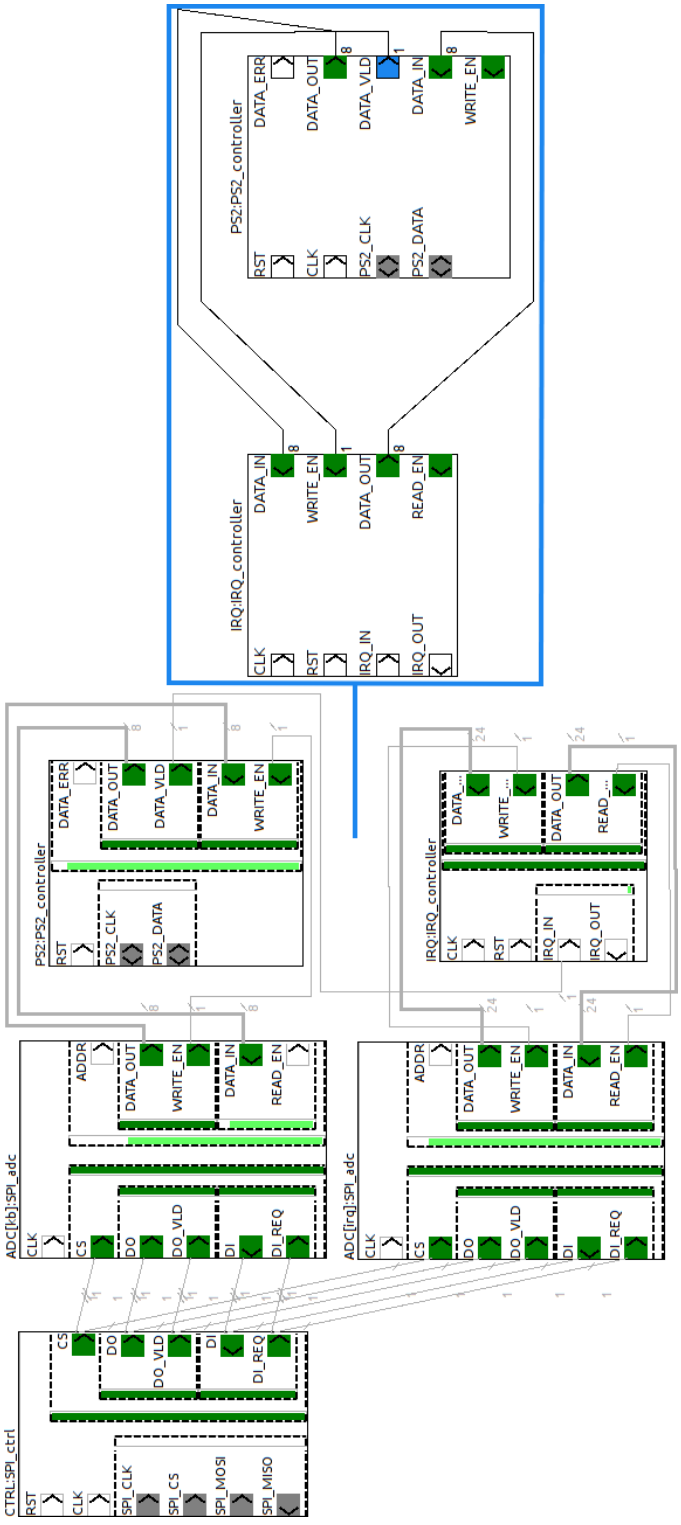

Obrázek B.1: Aplikace 1: Všechna propojení byla nalezena správně, až na propojení komponent Expander a LCD (v modrém rámečku je zachycen nejlépe ohodnocený výstup algoritmu).



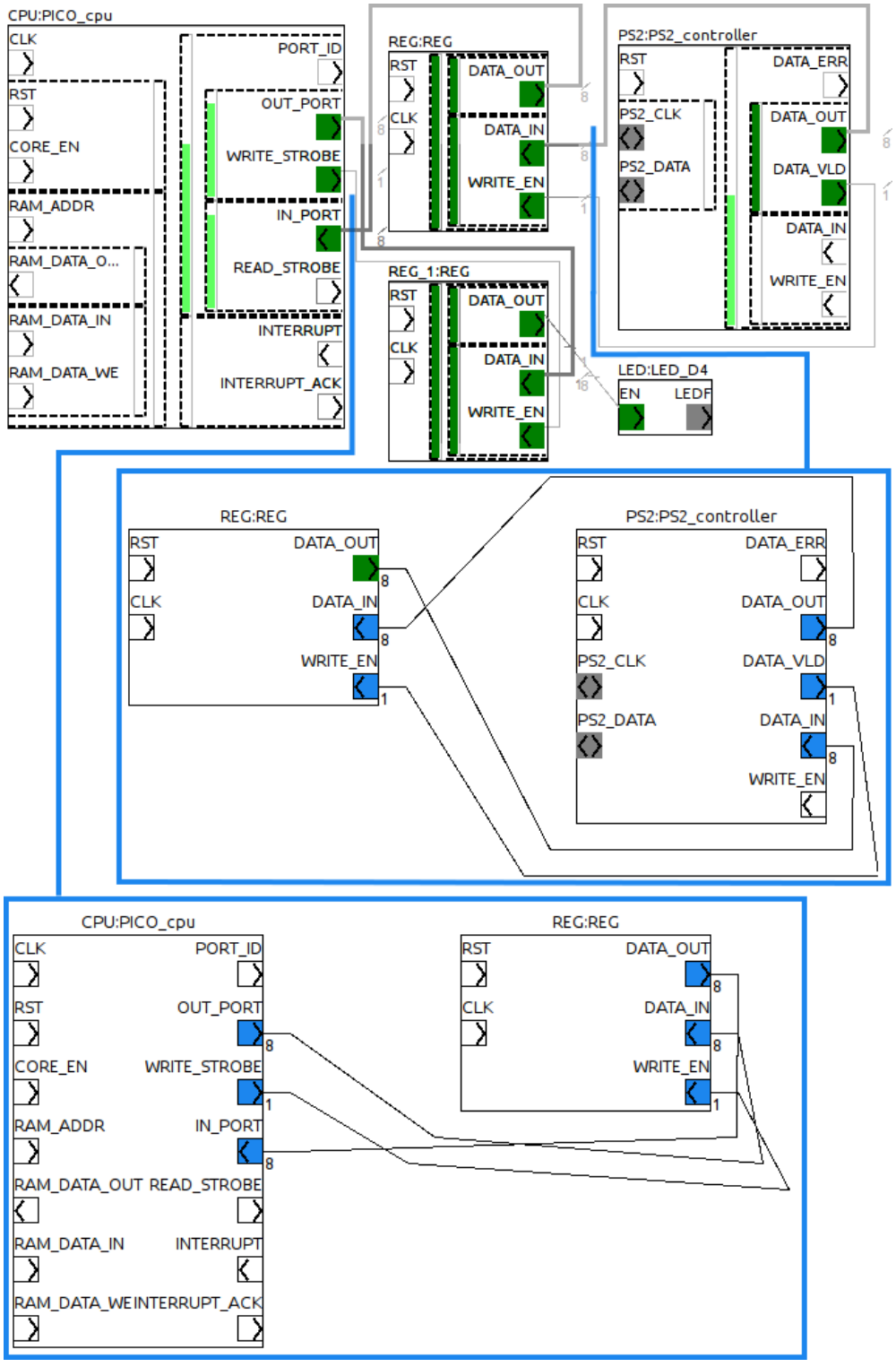

Obrázek B.2: Aplikace 2: Všechna propojení byla nalezena správně.



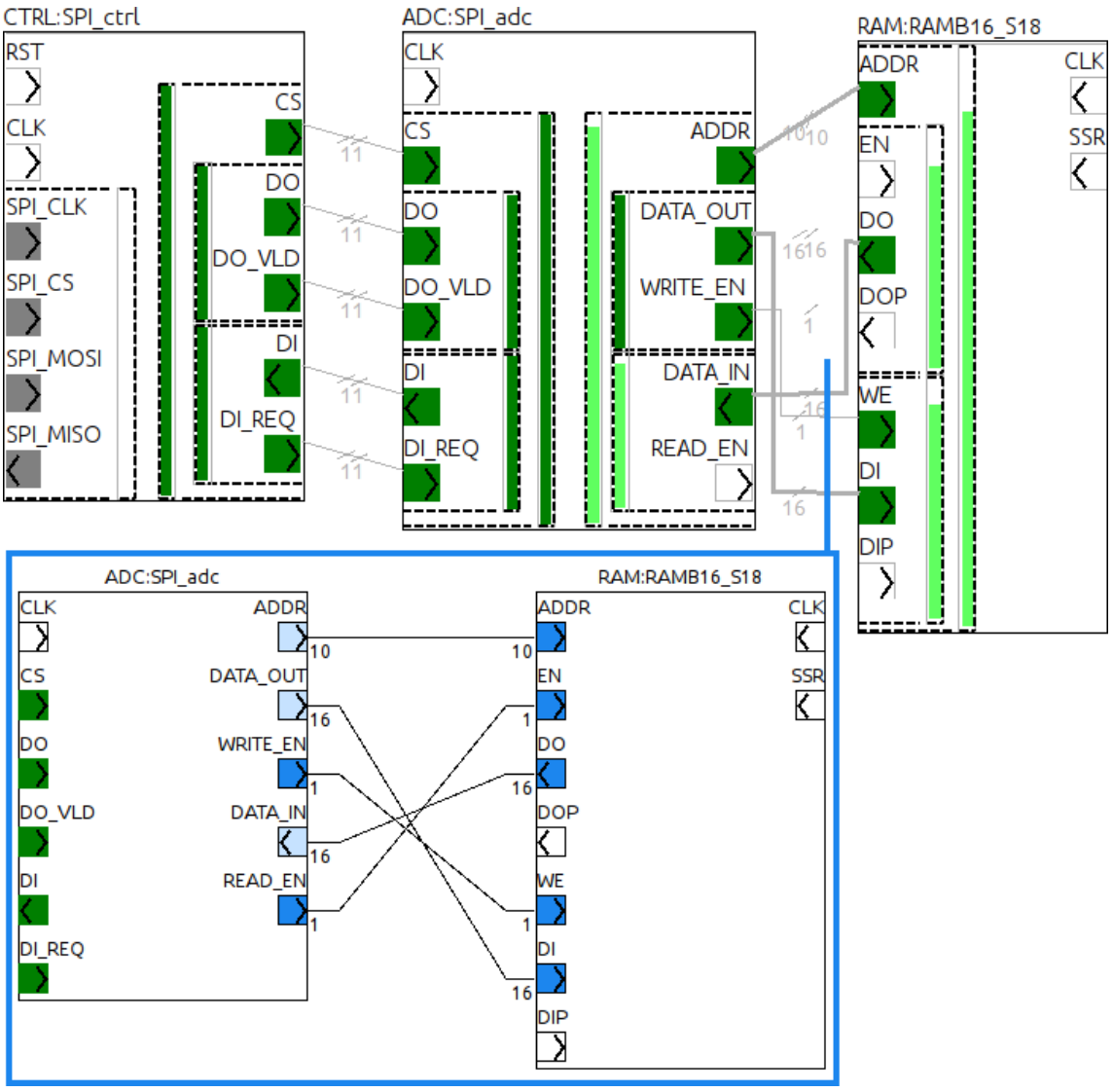

Obrázek B.3: Aplikace 3: Všechna propojení byla nalezena správně, až na propojení komponent KB_ctrl a IRQ (v modrém rámečku je zachycen nejlépe ohodnocený výstup algoritmu).




Obrázek B.4: Aplikace 4: Všechna propojení byla nalezena správně, až na propojení komponent PS2 a IRQ (v modrém rámečku je zachycen nejlépe ohodnocený výstup algoritmu).




Obrázek B.5: Aplikace 5: Zde bylo pouze jedno propojení nalezeno úplně správně (REG_1 a LED), v ostatních případech bylo hledané propojení vždy jen podmnožinou nejlepších nalezených (v modrých rámečcích jsou zachyceny nejlépe ohodnocené výstupy algoritmu).




Obrázek B.6: Aplikace 6: Zde nebylo správně nalezeno propojení komponent ADC a RAM, hledané propojení je pouze podmnožinou nalezeného propojení (v modrém rámečku je zachycen nejlépe ohodnocený výstup algoritmu).

# Příloha C

# Obsah CD

Na přiloženém CD se nachází:

- Tato technická zpráva ve formátu `pdf` a zdrojové soubory zprávy pro sázecí systém LaTeX (adresář `/zprava`).
- Zdrojový kód popsané implementace (adresář `/zdrojovy_kod`).
- Spustitelná instalace prostředí VLAM IDE včetně pracovního adresáře s aplikacemi z výuky, na kterých probíhalo popsané testování (adresář `/vlam_ide`).
- Programová dokumentace ke zdrojovým kódům implementace, která byla provedena v rámci této práce, vygenerovaná nástrojem `Doxygen` (adresář `/dokumentace`).
- Uživatelský manuál pro použití inferenčního algoritmu v prostředí VLAM IDE (adresář `/manual`).